

# Scheduling Network Maintenance Jobs with Release Dates and Deadlines to Maximize Total Flow Over Time: Bounds and Solution Strategies

Natashia Boland      Thomas Kalinowski      Simranjit Kaur

University of Newcastle, Australia

February 12, 2015

## Abstract

We consider a problem that marries network flows and scheduling, motivated by the need to schedule maintenance activities in infrastructure networks, such as rail or general logistics networks. Network elements must undergo regular preventive maintenance, shutting down the arc for the duration of the activity. Careful coordination of these arc maintenance jobs can dramatically reduce the impact of such shutdown jobs on the flow carried by the network. Scheduling such jobs between given release dates and deadlines so as to maximize the total flow over time presents an intriguing case to study the role of time discretization. Here we prove that if the problem data is integer, and no flow can be stored at nodes, we can restrict attention to integer job start times. However if flow can be stored, fractional start times may be needed. This makes traditional strong integer programming scheduling models difficult to apply. Here we formulate an exact integer programming model for the continuous time problem, as well as integer programming models based on time discretization that can provide dual bounds, and that can - with minor modifications - also yield primal bounds. The resulting bounds are demonstrated to have small gaps on test instances, and offer a good trade-off for bound quality against computing time.

**Keywords.** network models; maintenance scheduling; mixed integer programming; time discretization; heuristics

## 1 Introduction

We consider a problem of scheduling maintenance jobs on the arcs of a network, where each job has a given processing time, release date and deadline. The network is used to transport flow from a source node to a sink node using capacitated arcs, where the capacity of an arc represents the highest rate in flow units per unit time that it can carry. While a maintenance job is in progress on an arc, the arc cannot carry flow, and the objective is to schedule the maintenance jobs so as to maximize the total flow that can be transported by the network over a given planning time horizon.

This problem is a natural marriage of scheduling with network flow: it unites the diverse and well established field of scheduling (see any of the many excellent texts available on the topic, e.g. [1, 7, 8, 9, 13, 17, 19]) with dynamic network flows, which have been the subject of intense study in recent years, e.g. [11, 12, 21]. Yet the problem was only recently introduced, first appearing in [5], where meta-heuristic approaches were explored. It was motivated by a study of a bulk goods export supply chain [4, 6], in which maintenance jobs on sections of the rail network and equipment in the export terminals were scheduled so as to maximize the throughput of the system. Strong NP-hardness of the problem is established in [5] using a reduction to a network with only a single transshipment node (node other than the source or sink). The complexity of a variety of special cases is investigated in [3] where it is shown (among other results) that even in the case that all jobs have unit processing times and do not have release dates or deadlines, and that all transshipment nodes have equal inbound and outbound capacity, the problem is still strongly NP-hard.

The problem exhibits a rich structure, making it attractive for complexity analysis, approximation, combinatorial algorithms, integer programming, and heuristics. It represents a natural extension to existing

network models, and admits many interesting variants. For example, Tawarmalani and Li [22], motivated by a problem in highway maintenance, consider a multicommodity flow variant, providing complexity results, combinatorial algorithms, and integer programming models. The only other work combining network flow and scheduling that we are aware of is that of Nurre et al. [16], who schedule arc restoration tasks in the wake of a major disruption so as to maximize weighted flow over time. Whilst the problem does involve scheduling over a (weighted) maximum flow objective, there are several key differences to the problem we study here. In the problem of [16], the arc is closed from the start of the planning horizon until its restoration task is completed, from which time it is always open. The restoration task for each arc must be scheduled for completion by a work group, leading to a parallel machine scheduling structure absent in our case.

All work to date on the problem we study here has ignored the possibility of flow storage at nodes. The latter is an important feature of real applications: in the export supply chain studied in [4], the port terminal has stockyards for holding stockpiled material, which can provide outbound flow from the system even while maintenance shuts down inbound flow. The optimal schedule in the case that storage is ignored is likely to be far from optimal in the presence of storage. (We illustrate this point using the example given in Section 2.1.) Furthermore, the presence of storage has a fundamental effect on the properties of the problem: when no storage is allowed, integer data implies an optimal solution with integer job start times, but with storage, non-integer job start times may be required for optimality.

This makes the problem with storage a fascinating setting in which to study the role of time discretization in integer programming models. It motivates questions such as what potential start times are important to consider? Can we restrict attention to particular times, or must all times in the continuum be considered in order to prove optimality? What kinds of discretization and what models can guarantee valid bounds on the optimal solution? In addressing these questions, this paper can be viewed as contributing to a rapidly increasing body of work exploring exact models based on coarser time discretizations, for example big bucket models in lot-sizing [18] and discretizations based on job release dates and deadlines in machine scheduling [2], as well as approximate integer programs, for example for traveling salesman problems with time windows [10, 23] and for solving continuous-time dynamic network flows [14].

The prior work on close variants of the problem we study here either treats the selection of job start times heuristically, or assumes integer start times. In order to tackle a real problem in which maintenance jobs must be timed to within 15-minute intervals over a planning horizon of a year, the integer programming model presented in [4] is formulated in terms of a sparse set of possible start times selected heuristically for each job. The problem variants tackled in [3, 5] (all without storage) simply state the problem as one in which jobs must start at integer times. Whilst this property of start times is intuitively reasonable in the case without storage, it has not yet been formally proved; the first such proof is one of the key contributions of this paper. These are summarized as follows.

1. We give the first mixed integer linear programming model that solves the continuous time problem (with or without storage) exactly.
2. We provide the first formal proof that when no storage is allowed, there exists an optimal solution in which all job start times expressed in the form  $a + \varepsilon_1 b_1 + \varepsilon_2 b_2 + \dots + \varepsilon_k b_k$  where  $a$  is a release date or a deadline of some job,  $\varepsilon_i \in \{1, -1\}$  for all  $i \in \{1, \dots, k\}$ ,  $b_i$  is the processing time of some job for all  $i \in \{1, \dots, k\}$ , and the number of terms  $k$  is less than twice the total number of jobs. This yields the property that if the data is integer, an optimal solution with integer start times is assured.
3. We demonstrate that when storage *is* allowed, even with integer data, non-integer job start times may be needed in an optimal solution. We prove that in this case we can (without loss of generality) restrict attention to *rational* start times, and in particular show that there exist optimal start times that can be expressed as rational numbers with a denominator that is *independent* of the job parameters (processing times, release dates and deadlines).
4. As a consequence, the time indexed formulations so useful in machine scheduling (e.g. [20]) cannot be applied to this problem directly while assuring optimality. Noting that the exact formulation is difficult to solve in practice, we provide the first integer programming model to give valid *dual* bounds, (upper bounds), based on discretization of time. The model can employ *any* discretization. This property is particularly attractive, as it permits models to be time-scale invariant. For example, using the release dates and deadlines of jobs to form the time discretization yields a model with a size that is invariant

to the length of the time horizon, depending only on the number of jobs; a unit time discretization based model's size would increase with increasing time horizon as well as with the number of jobs.

5. We observe that the same model employing a discretization that conforms to the job parameters (release dates and deadlines are included, as is any time point in the discretization plus or minus any job processing time that stays between the job's release date and deadline) and with the additional restriction that jobs start at the start of a time interval, also provides feasible solutions when solved as an integer program (IP). Furthermore, if the time discretization is sufficiently fine, this formulation yields optimal solutions.
6. This allows us to compute primal (lower) bounds for test instances with integer data by solving an IP based on the unit time discretization. We compare this approach with several heuristics that "repair" solutions found in the process of solving the integer programs that yield dual bounds. Our computational tests demonstrate the strength of both lower and upper bounds; we discuss the trade-offs between solution time and quality, and show that for the best approaches, optimality gaps are typically very small.

## 2 Problem formulation

Let  $N = (V, A, s, t)$  be a network with node set  $V$ , arc set  $A$ , source node  $s \in V$ , and sink node  $t \in V$ . There is a set  $W \subseteq V \setminus \{s, t\}$  of storage nodes, and the capacity vector  $\mathbf{u} = (u_x)_{x \in A \cup W}$  collects the arc capacities and the storage capacities of the nodes in  $W$ . For a node  $v$ , let  $\delta^{\text{out}}(v)$  and  $\delta^{\text{in}}(v)$  denote the set of arcs starting at  $v$  and ending in  $v$ , respectively. In addition, we are given a set  $A_1 \subseteq A$  of arcs that need to be shut for a maintenance job to be done without any preemptions. For simplicity, assume that for each arc  $a \in A_1$  there is exactly one maintenance job, which is specified by its processing time  $p_a$ , its release time  $r_a$ , and its deadline  $d_a$ . We consider this network over a time horizon  $T$ . The problem is to schedule the jobs, hence a feasible solution is a vector  $\mathbf{t}^* = (t_a^*)_{a \in A_1}$  of start times with  $t_a^* \in [r_a, d_a - p_a]$  for all  $a \in A_1$ . Let  $X$  denote the set of all feasible solutions, i.e.  $X = \prod_{a \in A_1} [r_a, d_a - p_a]$ .

A feasible solution  $\mathbf{t}^*$  is evaluated as follows. Let  $0 = t_0 < t_1 < \dots < t_n = T$  be the increasing sequence obtained by ordering the set

$$\{t_a^* : a \in A_1\} \cup \{t_a^* + p_a : a \in A_1\} \cup \{0, T\}.$$

Note that the same time point  $t$  can occur multiple times in this union, since it is possible that several jobs start and end at time  $t$ . During each of the time intervals  $[t_{i-1}, t_i)$ ,  $i = 1, \dots, n$ , thus induced by  $\mathbf{t}^*$ , the state of the network is constant: no maintenance job either starts or ends within the interval.

For  $a \in A_1$ , let  $I_a$  denote the set of intervals in which arc  $a$  is shut for maintenance, i.e.,  $I_a = \{i \in \{1, \dots, n\} : t_a^* < t_i \leq t_a^* + p_a\}$ . The value of the solution  $\mathbf{t}^*$ , denoted by  $\text{val}(\mathbf{t}^*)$ , can be characterized as the optimal value of a maximum flow problem in a time-expanded network. For  $a \in A$  and  $i \in \{1, \dots, n\}$ , let  $x_{ai}$  be the flow on arc  $a$  in time interval  $[t_{i-1}, t_i)$ . For  $v \in W$  and  $i \in \{1, \dots, n\}$ , let  $x_{vi}$  be the amount of flow that is stored in  $v$  at time  $t_i$ . We impose the boundary conditions that the storage nodes are empty in the beginning and in the end of the time horizon, i.e.,  $x_{v0} = x_{vn} = 0$  for all  $v \in W$ . Finally,  $\text{val}(\mathbf{t}^*)$  is the

optimal objective value of the following problem.

$$\text{maximize } \sum_{i=1}^n \left( \sum_{a \in \delta^{\text{out}}(s)} x_{ai} - \sum_{a \in \delta^{\text{in}}(s)} x_{ai} \right) \quad (1)$$

$$\text{s.t. } \sum_{a \in \delta^{\text{out}}(v)} x_{ai} = \sum_{a \in \delta^{\text{in}}(v)} x_{ai} \quad i \in \{1, \dots, n\}, v \in V \setminus (W \cup \{s, t\}), \quad (2)$$

$$\sum_{a \in \delta^{\text{out}}(v)} x_{ai} + x_{vi} = \sum_{a \in \delta^{\text{in}}(v)} x_{ai} + x_{v,i-1} \quad i \in \{1, \dots, n\}, v \in W, \quad (3)$$

$$x_{ai} \leq (t_i - t_{i-1})u_a \quad a \in A \setminus A_1, i \in \{1, \dots, n\}, \quad (4)$$

$$x_{ai} \leq (t_i - t_{i-1})u_a \quad a \in A_1, i \in \{1, \dots, n\} \setminus I_a, \quad (5)$$

$$x_{ai} = 0 \quad a \in A_1, i \in I_a, \quad (6)$$

$$x_{vi} \leq u_v \quad v \in W, i \in \{0, 1, \dots, n\} \quad (7)$$

$$x_{v0} = x_{vn} = 0 \quad v \in W, \quad (8)$$

$$x_{ai}, x_{vi} \geq 0 \quad a \in A, v \in W, i \in \{1, \dots, n\}. \quad (9)$$

The objective function (1) is the total throughput, i.e., the sum of the flow values over all time periods. Constraints (2) and (3) are flow conservation constraints for non-storage nodes and storage nodes, respectively. The incoming flow of a storage node  $v \in W$  in time period  $i$ , i.e., in the time interval  $[t_{i-1}, t_i)$ , is the sum of the flow that arrives at node  $v$  in time period  $i$  and the flow that is stored in node  $v$  at time  $t_{i-1}$ . Similarly, the outgoing flow of a storage node  $v \in W$  in time period  $i$  is the sum of the flow that arrives at node  $v$  in this period and the flow that is stored in node  $v$  at time  $t_i$ . Constraints (4), (5) and (6) are capacity constraints, where (6) captures the arc outages. Constraints (7) are storage capacity constraints, and constraints (8) capture the boundary conditions. Our optimization problem is to find a start time vector  $\mathbf{t}^*$  to maximize the total throughput:

$$\varphi^* = \max\{\text{val}(\mathbf{t}^*) : \mathbf{t}^* \in X\}. \quad (10)$$

Note that we may assume that in each interval  $[t_{i-1}, t_i)$  the flow rates for all arcs are constant: on arc  $a$ , we have a flow rate of  $x_{ai}/(t_i - t_{i-1})$  units of flow per time unit.

## 2.1 An illustrative example

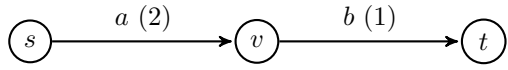
Both to illustrate the problem, and to demonstrate the importance of storage in finding an optimal maintenance schedule, we consider the network and arc maintenance jobs given in Figure 1, where arc labels indicate arc names and capacities (in parentheses). In this example, the time horizon is  $T = 3$ , and only the job on arc  $a$  needs to be scheduled: the job on arc  $b$  must start at time  $t_b^* = 0$ . Let  $\hat{t}$  denote the start time of the job on arc  $a$ . From job  $a$ 's parameters, we see that  $\hat{t} \in [0, 1]$ . As a consequence, there can only be *one* sequence of times  $0 = t_0 \leq t_1 \leq t_2 \leq \dots \leq t_n = T$  resulting from ordering  $\{\hat{t}, t_b^*\} \cup \{\hat{t} + p_a, t_b^* + p_b\} \cup \{0, T\} = \{0, \hat{t}, 1, \hat{t} + 2, 3\}$ , (where we have relaxed the strict inequalities to permit the job on  $a$  to start at  $t_0 = 0$  or at  $t_b^* + p_b = 1$ , the end time of the job on  $b$ , or to end at time  $T = 3$ ), and that is given by

$$0 = t_0 \leq t_1 = \hat{t} \leq 1 = t_2 < t_3 = \hat{t} + 2 \leq t_4 = T = 3.$$

The four time intervals thus induced have duration  $\hat{t}, 1 - \hat{t}, \hat{t} + 1, 1 - \hat{t}$ , respectively, and the corresponding flow networks for no storage and for storage at  $v$  with capacity 2 are illustrated in Figure 2. For the no-storage case the maximum throughput is

$$0 + 0 + 0 + \min\{2 - 2\hat{t}, 1 - \hat{t}\} = 1 - \hat{t}.$$

This is maximized for  $\hat{t} = 0$ , i.e., the optimal schedule is to take  $t_a^* = t_b^* = 0$ , which gives a total throughput of 1. Note that setting  $\hat{t} = 1$  gives a solution with zero throughput. For the case with storage, the total capacity of arc  $b$  gives an upper bound of  $(\hat{t} + 1) + (1 - \hat{t}) = 2$  for the throughput, and this bound can be achieved only if arc  $b$  is at capacity in each time period. Therefore it is necessary that  $2\hat{t} \geq \hat{t} + 1$ , i.e.,  $\hat{t} \geq 1$ .



Network

arc	$r$	$d$	$p$
$a$	0	3	2
$b$	0	1	1

Job parameters

Figure 1: Example to show the effect of storage capacity on the optimal maintenance schedule.

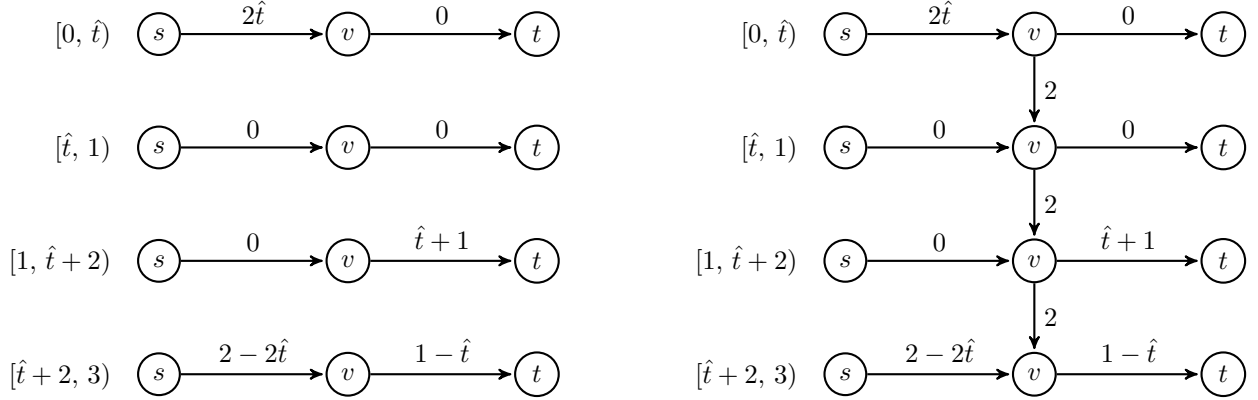


Figure 2: Time expanded networks without storage (left) and with storage (right). The capacity of each arc during the time interval indicated at the left of the network is written above the arc.

This is also sufficient, since for  $\hat{t} = 1$  we get a feasible solution of value 2 which is shown on the left in Figure 3. The situation is precisely the reverse of the case without storage: for  $\hat{t} = 0$  we obtain the smallest possible objective value, namely 1 (see the right hand side solution in Figure 3).

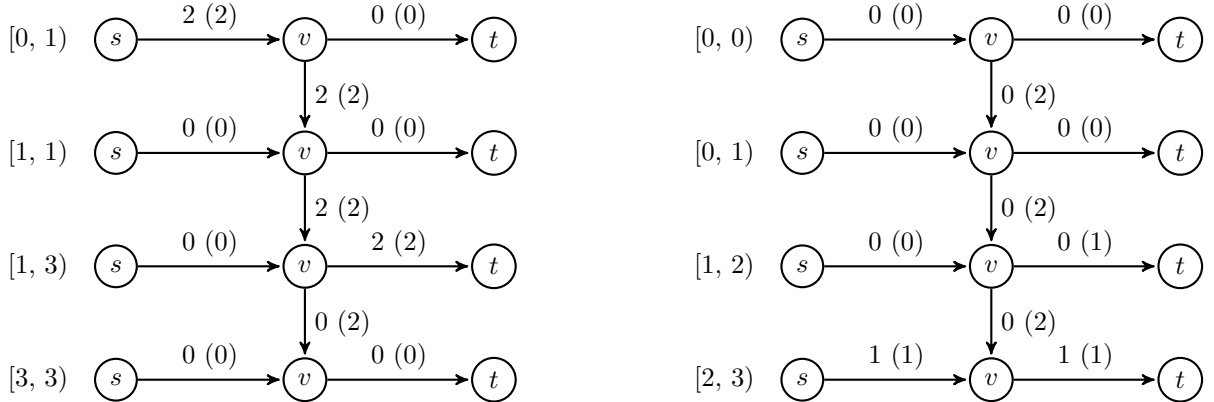


Figure 3: Optimal flows for  $\hat{t} = 1$  (left) and  $\hat{t} = 0$  (right) in the case with storage. The arcs are labeled with flow values and, in parentheses, capacities, during the time interval indicated at the left of the network.

This example shows that ignoring storage can in some sense be as bad as can be: the optimal schedule without storage is exactly the schedule that minimizes flow if storage is allowed, and vice versa.

## 2.2 A mixed integer linear programming formulation

We first model the problem as a nonlinear mixed integer program and then linearise the nonlinear constraints with the help of additional variables. Since we need at most two time points for each job to indicate its start and end time, as well as the time horizon start and end times, we use variables  $t_0, t_1, \dots, t_{M-1}, t_M$  where

$M = 2|A_1| + 1$ . Clearly we require

$$0 = t_0 \leq t_1 \leq \dots \leq t_{M-1} \leq t_M = T. \quad (11)$$

We introduce binary variable  $w_{ai} = 1$  if arc  $a \in A_1$  is undergoing maintenance (i.e. is *not* available) in time interval  $[t_{i-1}, t_i]$  and zero otherwise. For convenience in what follows, we include  $w_{a0} = 0$  for each  $a \in A_1$ .

If  $w_{ai} = 1$  then the start time of interval  $i$  must be at or after the release date of the job on  $a$ . The implication  $w_{ai} = 1 \implies t_{i-1} \geq r_a$  can be modelled linearly with

$$t_{i-1} \geq r_a w_{ai}, \quad \text{for all } i \in \{1, \dots, M\}, a \in A_1. \quad (12)$$

Similarly, if  $w_{ai} = 1$  then the end time of interval  $i$  must be at or before the due date of the job on  $a$ , i.e., we want to model the implication  $w_{ai} = 1 \implies t_i \leq d_a$ , which can be done by the linear constraint

$$t_i \leq d_a + (T - d_a)(1 - w_{ai}), \quad \forall i \in \{1, \dots, M\}, a \in A_1. \quad (13)$$

This constraint becomes  $t_i \leq d_a$  when  $w_{ai} = 1$ , and  $t_i \leq T$  when  $w_{ai} = 0$ .

To ensure that the arc is shut precisely for the requisite duration, we constrain the total duration of the time intervals in which the arc is undergoing maintenance to equal the maintenance job processing time. This can be enforced by the nonlinear constraint

$$\sum_{i=1}^M (t_i - t_{i-1}) w_{ai} = p_a, \quad \forall a \in A_1. \quad (14)$$

To enforce job processing without preemption, we require that each of the vectors  $(w_{ai})_{i=0,1,\dots,M}$  has the consecutive ones property: it consists of a sequence of 0's, then a sequence of 1's, then a sequence of 0's. To enforce this property, we introduce another binary variable:  $z_{ai} = 1$  if the processing of the job on arc  $a \in A_1$  begins at the beginning of interval  $i$ . The following two constraints ensure that the maintenance job on an arc begins exactly once in the time horizon and once the arc is shut for maintenance then it remains so for consecutive time intervals:

$$\sum_{i=1}^M z_{ai} = 1, \quad a \in A_1, \quad (15)$$

and

$$z_{ai} \geq w_{ai} - w_{a(i-1)}, \quad \forall i \in \{1, \dots, M\}, a \in A_1. \quad (16)$$

To ensure flow cannot pass through the arc while it is shut, we use another nonlinear constraint:

$$x_{ai} \leq (t_i - t_{i-1})(1 - w_{ai})u_a, \quad \forall i \in \{1, \dots, M\}, a \in A_1. \quad (17)$$

The resulting formulation, which we refer to as the *Continuous Time Nonlinear Integer Program*, denoted by CTIP-NL, is given by

$$\begin{aligned} & \text{maximize} \quad \sum_{i=1}^M \left( \sum_{a \in \delta^{\text{out}}(s)} x_{ai} - \sum_{a \in \delta^{\text{in}}(s)} x_{ai} \right) \\ & \text{s.t.} \quad \sum_{a \in \delta^{\text{out}}(v)} x_{ai} = \sum_{a \in \delta^{\text{in}}(v)} x_{ai} & i \in \{1, \dots, M\}, v \in V \setminus (W \cup \{s, t\}), \\ & \quad \sum_{a \in \delta^{\text{out}}(v)} x_{ai} + x_{vi} = \sum_{a \in \delta^{\text{in}}(v)} x_{ai} + x_{v,i-1} & i \in \{1, \dots, M\}, v \in W, \\ & \quad x_{ai} \leq (t_i - t_{i-1})u_a & i \in \{1, \dots, M\}, a \in A \setminus A_1, \\ & \quad 0 \leq x_{vi} \leq u_v & i \in \{0, 1, \dots, M\}, v \in W, \\ & \quad x_{v0} = x_{vM} = 0 & v \in W, \\ & \quad w_{a0} = 0 & a \in A_1, \\ & \quad (11), (12), (13), (15), (16), (14), (17), \\ & \quad x_{ai} \geq 0 & i \in \{1, \dots, M\}, a \in A, \\ & \quad w_{ai} \in \{0, 1\} & i \in \{0, 1, \dots, M\}, a \in A_1, \text{ and} \\ & \quad z_{ai} \in \{0, 1\} & i \in \{1, \dots, M\}, a \in A_1. \end{aligned} \quad (18)$$

The nonlinear constraints in the above formulation can readily be modelled linearly with the use of additional variables. We define  $\Delta_{ai}$  and  $\bar{\Delta}_{ai}$  for each  $a \in A_1$  and  $i \in \{1, \dots, M\}$  by

$$\Delta_{ai} = \begin{cases} t_i - t_{i-1}, & \text{if } w_{ai} = 1 \\ 0, & \text{otherwise} \end{cases} \quad \text{and} \quad \bar{\Delta}_{ai} = \begin{cases} t_i - t_{i-1}, & \text{if } w_{ai} = 0 \\ 0, & \text{otherwise.} \end{cases}$$

This can be modelled linearly via the constraints

$$\Delta_{ai} + \bar{\Delta}_{ai} = t_i - t_{i-1}, \quad \forall a \in A_1, i \in \{1, \dots, M\}, \quad (19)$$

together with

$$\Delta_{ai} \leq p_a w_{ai}, \quad \forall a \in A_1, i \in \{1, \dots, M\}, \quad (20)$$

and

$$\bar{\Delta}_{ai} \leq (T - p_a)(1 - w_{ai}), \quad \forall a \in A_1, i \in \{1, \dots, M\}. \quad (21)$$

Then (14) and (17) can be modelled linearly with

$$\sum_{i=1}^M \Delta_{ai} = p_a, \quad \forall a \in A_1, \quad (22)$$

and

$$x_{ai} \leq \bar{\Delta}_{ai} u_a, \quad \forall i \in \{1, \dots, M\}, a \in A_1. \quad (23)$$

We refer to the mixed integer linear programming formulation obtained by adding constraints (19), (20), and (21) to CTIP-NL and replacing the nonlinear constraints (14) and (17) with (22) and (23) respectively in CTIP-NL as the *Continuous Time Integer Program*, denoted by CTIP.

### 3 Properties of optimal solutions

In this section we derive properties of optimal solutions for both the cases: a) when there are no storage nodes, i.e.  $W = \emptyset$ , and b) when there are storage nodes, i.e.  $W \neq \emptyset$ .

For the case when there are no storage nodes, i.e.,  $W = \emptyset$ , we will prove in Lemma 2 that, without loss of generality, we may assume finitely many possible start times for each job. For each arc  $a \in A_1$  we will construct a finite subset  $S(a) \subseteq [r_a, d_a - p_a]$ , and then prove that there is always an optimal solution  $\mathbf{t}^*$  such that  $\mathbf{t}_a^* \in S(a)$  for every  $a \in A_1$ . Clearly we should consider to start the job on arc  $a$  as early as possible or as late as possible, so the set  $S_0(a) = \{r_a, d_a - p_a\}$  should be contained in  $S(a)$  for every  $a \in A_1$ . Now suppose that the job on arc  $a'$  starts at time  $t$ . If  $t < r_a + p_a$  and  $t + p_{a'} < d_a - p_a$ , then we might consider to start the job on arc  $a$  at time  $t + p_{a'}$  in order to start the job on arc  $a$  as early as possible while avoiding overlap of the two jobs. By similar reasoning it can be reasonable to start the job on arc  $a$  at time  $t$ , at time  $t - p_a$  or at time  $t + p_{a'} - p_a$  if these are in  $[r_a, d_a - p_a]$ . So we get a new candidate start time set  $S_1(a)$  for arc  $a$  by adding to  $S_0(a)$  all times in  $[r_a, d_a - p_a]$  that can be written in the form  $t$ ,  $t + p_{a'}$ ,  $t - p_a$ , or  $t + p_{a'} - p_a$  for some  $t \in S_0(a')$ . Assuming that we have already defined candidate start time sets  $S_k(a)$  for some nonnegative integer  $k$  and all  $a \in A_1$ , we can extend these sets to sets  $S_{k+1}(a)$  in the same way. More formally, for a set  $X \subseteq \mathbb{R}$  and a real number  $\lambda$  we write  $X + \lambda$  for the set  $\{x + \lambda : x \in X\}$ , and we define recursively,

$$S_{k+1}(a) = S_k(a) \cup \bigcup_{a' \in A_1 \setminus \{a\}} \left( (S_k(a') \cup (S_k(a') + p_{a'}) \cup (S_k(a') - p_a) \cup (S_k(a') + p_{a'} - p_a) \right) \cap [r_a, d_a - p_a], \quad (24)$$

and finally,

$$S(a) = S_{|A_1|-1}(a). \quad (25)$$

We claim that in order to solve (10) it is sufficient to maximize over the finite set  $X' = \prod_{a \in A_1} S(a)$ . Suppose we have an optimal solution  $\mathbf{t}^*$  and assume  $A_0 = \{a \in A_1 : \mathbf{t}_a^* \notin S(a)\} \neq \emptyset$ . Our argument is based on

performing a sequence of modification steps on this solution without losing optimality. In order to describe the single modification step we define a graph on the vertex set  $A_1$  associated with the current solution: Two arcs  $a, a' \in A_1$  are joined by an edge if  $\{t_a^*, t_a^* + p_a\} \cap \{t_{a'}^*, t_{a'}^* + p_{a'}\} \neq \emptyset$ . For  $a \in A_1$  denote the connected component of  $a$  in this graph by  $\mathcal{C}(a)$ , and let  $h$  be the distance function for this graph:  $h(a, a')$  is the minimal length of a path from  $a$  to  $a'$  if such a path exists, and  $\infty$  otherwise.

**Lemma 1.** *For  $a \in A_0$  and  $a' \in \mathcal{C}(a)$ ,  $t_{a'}^* \notin S_{|A_1| - h(a, a') - 1}(a')$ . In particular,  $t_{a'}^* \notin S_0(a')$  for  $a' \in \mathcal{C}(a)$ .*

*Proof.* Let  $h = h(a, a')$  and assume  $t_{a'}^* \in S_{|A_1| - h - 1}(a')$ . Let  $a' = a_0, a_1, \dots, a_h = a$  be a minimal path from  $a'$  to  $a$ . Using (24) we deduce  $t_{a_i}^* \in S'_{|A_1| - h - 1 + i}(a')$  for  $i = 1, 2, \dots, h$ . For  $i = h$  this is  $t_a^* \in S'_{|A_1| - 1}(a)$ , contradicting the hypothesis  $a \in A_0$ .  $\square$

**Lemma 2.** *If  $W = \emptyset$ , then there is an optimal solution for (10) such that  $t_a^* \in S(a)$  for each  $a \in A_1$ .*

*Proof.* By Lemma 1, no job on any arc  $a'$  in the component of an arc  $a \in A_0$  starts at one of its boundary start times  $r_{a'}$  or  $d_{a'} - p_{a'}$ . So we can shift all jobs in  $\mathcal{C}(a)$  by  $\pm \varepsilon$  for some  $\varepsilon > 0$  without becoming infeasible and without changing the graph associated with the solution. By optimality, such a shift (in either direction) leaves the objective value unchanged (see Lemma 3 in the appendix). Let  $\mathbf{t}^*(\varepsilon)$  be the solution obtained from  $\mathbf{t}^*$  by shifting the jobs in  $\mathcal{C}(a)$  by  $\varepsilon$  to the right, i.e.,

$$t^*(\varepsilon)_{a'} = \begin{cases} t_{a'}^* + \varepsilon & \text{for } a' \in \mathcal{C}(a), \\ t_{a'}^* & \text{for } a' \in A_1 \setminus \mathcal{C}(a). \end{cases}$$

Pick the smallest value  $\varepsilon$  such that for the solution  $\mathbf{t}^*(\varepsilon)$  we have

- $\{t^*(\varepsilon)_{a'}, t^*(\varepsilon)_{a'} + p_{a'}\} \cap \{t^*(\varepsilon)_{a''}, t^*(\varepsilon)_{a''} + p_{a''}\} \neq \emptyset$  for some  $a' \in \mathcal{C}(a)$  and  $a'' \in A_1 \setminus \mathcal{C}(a)$ , or
- $t^*(\varepsilon)_{a'} = d_{a'} - p_{a'}$  for some  $a' \in \mathcal{C}(a)$ .

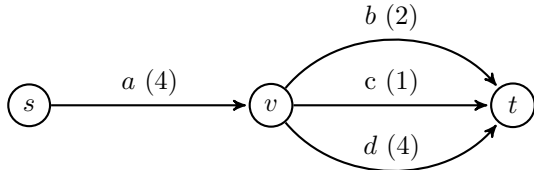
By construction, the graph corresponding to the new solution  $\mathbf{t}^*(\varepsilon)$  has more edges than the graph for the original solution, or the size of  $A_0$  decreases. Hence, this modification can be iterated only a finite number of times, and this iterated process terminates with an optimal solution with  $A_0 = \emptyset$ .  $\square$

By Lemma 2, there is an optimal solution in the finite set  $X'$ , and the proof implies the following integrality property.

**Proposition 1.** *If the input data is integer and  $W = \emptyset$ , then there is an integral optimal solution, i.e., an optimal solution  $\mathbf{t}^*$  with  $t_a^* \in \mathbb{Z}$  for all  $a \in A_1$ .*

The statement of Proposition 1 is in general not true for the problem involving storage, as is indicated by the following example.

**Example 1.** Consider the network in Figure 4 over a time horizon  $T = 7$ , and suppose that node  $v$  has storage capacity 3 and we have four jobs with parameters shown in the table in Figure 4.



Network

arc	$r$	$d$	$p$
$a$	0	5	3
$b$	3	5	2
$c$	0	5	5
$d$	0	6	6

Job parameters

Figure 4: Network and job parameters for Example 1.

The job on arc  $a$  is the only job that can be moved, and its start time  $\hat{t}$  has to be in the interval  $[0, 2]$ . The time slicing is given by



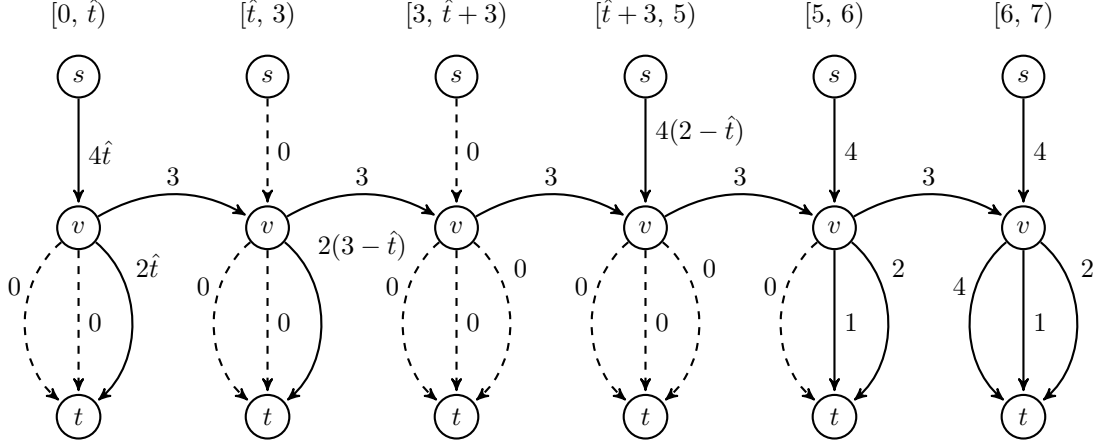


Figure 5: The time-expanded networks for the instance in Example 1. Arcs under maintenance are indicated by dashed lines.

$$(t_0, t_1, \dots, t_6) = (0, \hat{t}, 3, \hat{t} + 3, 5, 6, 7)$$

and the corresponding time-expanded network is shown in Figure 5. The total capacity of the arcs going into  $t$  is

$$2\hat{t} + 2(3 - \hat{t}) + 1 + 2 + 4 + 1 + 2 = 16.$$

A total throughput of 16 can be achieved if and only if there is a feasible solution in which all the arcs into node  $t$  are at capacity. In order for arc  $b$  to be at capacity in the first two time periods, it is necessary that the flow on arc  $a$  in the first time period is at least 6, which implies  $4\hat{t} \geq 6$ , i.e.,  $\hat{t} \geq 3/2$ . On the other hand, the total capacity of the arcs out of node  $s$  is

$$4\hat{t} + 4(2 - \hat{t}) + 4 + 4 = 16.$$

So in order to achieve a total throughput of 16 the arcs out of node  $s$  have to be at capacity in each time period as well. For the first time period, this implies  $4\hat{t} - 2\hat{t} \leq 3$ , and therefore  $\hat{t} \leq 3/2$ . We conclude that for a total throughput of 16 it is necessary that the job on arc  $a$  starts at time  $3/2$ . This is also sufficient, as can be seen from the solution illustrated in Figure

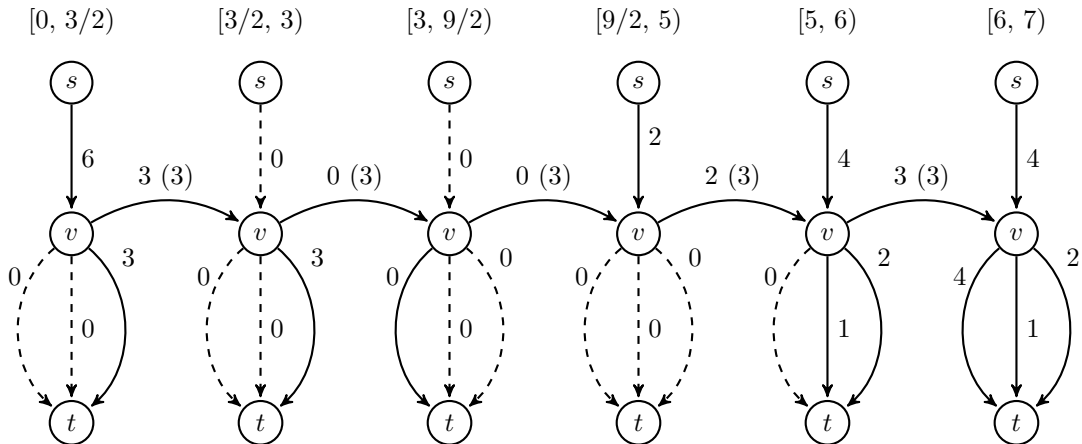


Figure 6: The flow of value 16 for  $\hat{t} = 3/2$ . All arcs except the arcs between copies of the storage node  $v$  are at capacity.

In the following proposition we prove that if there are storage nodes, i.e.  $W \neq \emptyset$ , then we can restrict our attention to rational start times for jobs.

**Proposition 2.** *If  $W \neq \emptyset$ , then there exists an optimal solution in which all jobs start at rational times. Further, the denominators of these optimal rational start times do not depend on parameters of the jobs (processing times, release dates or deadlines).*

*Proof.* Observe that the constraint matrix and the right hand side vector for the CTIP formulation contain only integer entries. So by the fundamental theorem for Mixed Integer Linear Programs [15] the convex hull of the set of feasible solutions for CTIP is a rational polyhedron, with all its extreme points rational. Hence there exists an optimal solution with rational start times.

Now this optimal solution of CTIP is obtained at some leaf node of the corresponding branch and bound tree. At this leaf node, all binary variables in CTIP are fixed, and the start times of the jobs are obtained by solving the resulting linear programming formulation so as to yield an extreme point (basic feasible) solution. Such an extreme point has the form  $B^{-1}b$ , where  $B$  is a square nonsingular submatrix of the constraint matrix and  $b$  is a corresponding vector of the constant terms in the constraints. By Cramer's rule,  $B^{-1} = \frac{1}{|B|}B^*$  where  $B^*$  and  $|B|$  are the adjoint and determinant of the matrix  $B$  respectively. For integer data, the entries of  $B^*$  and  $b$  are integer, as is  $|B|$ . So the (rational) solution to the linear program consists of integer multiples of  $\frac{1}{|B|}$ . Now the claim follows by observing that once all binary variables are fixed in CTIP, the entries in the constraint matrix of the remaining linear program are 0,  $\pm 1$ , or an element of  $\{\pm u_a : a \in A\}$ , and all entries of the right hand side vector are integers. In particular, the only constraints with coefficients *not* in  $\{-1, 0, 1\}$  are (18) and (23).  $\square$

Although the exact continuous time formulation CTIP has allowed us to establish the above important property of the problem with storage, it unfortunately performs very poorly in practice (as we shall see in Section 6). Thus other, more efficient, approaches to solving the problem are of interest. In the next section we consider such approaches to finding upper bounds, and in the subsequent section consider lower bounds.

## 4 Upper bounds

In this section we present an approximate MIP model which gives upper bounds for the problem. This upper bound model can be obtained from any given discretization  $0 = t_0 < t_1 < \dots < t_n = T$  of the time horizon. Importantly, here the  $(t_i)_{i=0, \dots, n}$  are fixed input parameters, not decision variables as they were in the CTIP model, and  $n$  is an arbitrary (given) positive integer. For  $i \in \{1, \dots, n\}$ , we refer to  $[t_{i-1}, t_i)$  as interval  $i$ . For  $a \in A_1$ , let  $\mathcal{S}_a$  be the set of intervals in which job  $a$  can start, and let  $\mathcal{T}_a$  be the set of intervals in which job  $a$  can be partially processed. More precisely,  $\mathcal{S}_a = \{i : r_a < t_i, d_a - p_a \geq t_{i-1}\}$  and  $\mathcal{T}_a = \{i : d_a > t_{i-1}, r_a < t_i\}$ . Our upper bound model has the following variables:

- $x_{ai}$ : total flow on arc  $a$  in interval  $i$  ( $a \in A, i \in \{1, \dots, n\}$ ),
- $x_{vi}$ : amount of flow stored in node  $v$  at time  $t_i$  ( $v \in W, i \in \{1, \dots, n\}$ ),
- $y_{ai}$ : binary start indicator, i.e.,  $y_{ai} = 1 \iff t_{i-1} \leq t_a^* < t_i$  ( $a \in A_1, i \in \mathcal{S}_a$ ), and
- $z_{ai}$ : fraction of interval  $i$  for which job  $a$  is processed ( $a \in A_1, i \in \mathcal{T}_a$ ).

To ensure that every job is processed exactly once, we have the constraints

$$\sum_{i \in \mathcal{S}_a} y_{ai} = 1 \quad \forall a \in A_1. \quad (26)$$

Clearly, the job on an arc  $a \in A_1$  cannot be processed for more than  $\min\{t_i, d_a\} - \max\{t_{i-1}, r_a\}$  time in an interval  $i \in \mathcal{T}_a$ . So we add the constraints

$$(t_i - t_{i-1})z_{ai} \leq \min\{t_i, d_a\} - \max\{t_{i-1}, r_a\} \quad \forall a \in A_1, i \in \mathcal{T}_a. \quad (27)$$

The processing times are enforced by the constraints

$$\sum_{i \in \mathcal{T}_a} (t_i - t_{i-1})z_{ai} = p_a \quad \forall a \in A_1. \quad (28)$$

Next we describe how the  $z$  variables and the  $y$  variables are linked. For  $a \in A_1$  and  $i \in \mathcal{S}_a$ , let  $Q_{ai} \subseteq \mathcal{T}_a$  be the set of intervals that can be affected by job  $a$  when it starts in interval  $i$ , i.e.,

$$Q_{ai} = \{k \in \mathcal{T}_a : k \geq i, t_i + p_a > t_{k-1}\}.$$

When job  $a$  starts in interval  $i$ , then it has to be completed within  $Q_{ai}$ , which gives the constraints

$$\sum_{k \in Q_{ai}} (t_i - t_{i-1}) z_{ak} \geq p_a y_{ai} \quad \forall a \in A_1, i \in \mathcal{S}_a. \quad (29)$$

In the opposite direction, let  $P_{ai}$  be the set of intervals  $k$  such that starting job  $a$  in interval  $k$  can affect interval  $i$ . Let  $P_{ai}^*$  be the subset of these intervals  $k$  such that starting job  $a$  in interval  $k$  closes arc  $a$  for the whole interval  $i$  or until completion of job  $a$ . More precisely,

$$\begin{aligned} P_{ai} &= \{k \in \mathcal{S}_a : t_{k-1} \leq t_{i-1} < t_k + p_a\}, \\ P_{ai}^* &= \{k \in P_{ai} : \max\{t_{k-1}, r_a\} + p_a \geq \min\{t_i, d_a\}\}. \end{aligned}$$

For  $k \in P_{ai}$ , let  $\mu_{aki}^+$  and  $\mu_{aki}^-$  be upper and lower bounds for  $(t_i - t_{i-1}) z_{ai}$  if job  $a$  starts in interval  $k$ , i.e.,

$$\begin{aligned} \mu_{aki}^+ &= \min\{t_i, d_a, t_k + p_a\} - \max\{t_{i-1}, r_a\} \\ \mu_{aki}^- &= \begin{cases} \max\{0, \min\{t_k, d_a\} - (d_a - p_a)\} & \text{for } k = i, i \in \mathcal{S}_a, \\ \min\{t_i, d_a\} - \max\{t_{i-1}, r_a\} & \text{for } k \in P_{ai}^* \setminus \{i\}, \\ \max\{0, \max\{t_{k-1}, r_a\} + p_a - t_{i-1}\} & \text{for } k \in P_{ai} \setminus (P_{ai}^* \cup \{i\}). \end{cases} \end{aligned}$$

We add the constraints

$$\sum_{k \in P_{ai}} \mu_{aki}^- y_{ak} \leq (t_i - t_{i-1}) z_{ai} \leq \sum_{k \in P_{ai}} \mu_{aki}^+ y_{ak} \quad \forall a \in A_1, i \in \mathcal{T}_a. \quad (30)$$

The arc capacities are described by

$$x_{ai} \leq (t_i - t_{i-1})(1 - z_{ai}) u_a \quad \forall a \in A_1, i \in \mathcal{T}_a, \quad (31)$$

$$x_{ai} \leq (t_i - t_{i-1}) u_a \quad \forall a \in A_1, i \in \{1, \dots, n\} \setminus \mathcal{T}_a. \quad (32)$$

Finally adding the flow conservation constraints (2) and (3), the arc capacity constraints (4) for the arcs in the set  $A \setminus A_1$ , together with the storage node capacity constraints (7) and (8), the upper bound for (10) associated with the given time discretization is

$$\varphi = \max_{(\mathbf{x}, \mathbf{y}, \mathbf{z}) \in F} \left\{ \sum_{i=1}^n \left( \sum_{a \in \delta^{\text{out}}(s)} x_{ai} - \sum_{a \in \delta^{\text{in}}(s)} x_{ai} \right) : (2), (3), (4), (7), (8), (26)-(32) \right\} \quad (33)$$

where

$$F = \mathbb{R}_{\geq 0}^{(|A|+|W|)n} \times \{0, 1\}^{\sum_{a \in A_1} |\mathcal{S}_a|} \times [0, 1]^{\sum_{a \in A_1} |\mathcal{T}_a|}.$$

We refer to this formulation as the *Time Discretized Integer Program*, denoted by TDIP. Since it consists only of constraints that must be satisfied by any feasible solution, we have the following result.

**Proposition 3.** *Let  $0 = t_0 < t_1 < \dots < t_n = T$  be any time discretization. Then  $\varphi^* \leq \varphi$ , where  $\varphi^*$  is the optimal value of the original problem (10) and  $\varphi$  is the optimal value of (33).*

In Section 6, we experiment with two variants of TDIP. The time indexed model TDIP(TI) refers to unit time discretization, i.e.,  $n = T$  and  $t_i = i$  for  $i = 0, 1, \dots, T$ , while the release date/deadline model TDIP(RD) refers to the discretization consisting of the release times and deadlines of all jobs, i.e. the time discretization with  $\{t_i : i = 0, 1, \dots, n\}$  equal to the set

$$\mathcal{D} := \{r_a : a \in A_1\} \cup \{d_a : a \in A_1\} \cup \{0, T\}.$$

## 5 Lower bounds

If the same model employs a discretization  $0 = t_0 < t_1 < \dots < t_n = T$  with the property that

- (i)  $\bigcup_{a \in A_1} \{r_a, d_a\} \subseteq \Gamma \stackrel{\text{def}}{=} \{t_0, t_1, \dots, t_n\}$ , and
- (ii) for any  $i \in \{0, 1, \dots, n\}$  and any  $a \in A_1$ , if  $t_i \in [r_a, d_a - p_a]$  then  $t_i + p_a \in \Gamma$  and if  $t_i \in [r_a + p_a, d_a]$  then  $t_i - p_a \in \Gamma$ ,

then with minor modifications to some model parameters and the additional restriction that  $z_{ai} \in \{0, 1\}$  for all  $a \in A_1$ ,  $i \in \mathcal{T}_j$ , any feasible solution to the model will provide a lower bound on the optimal value of the original problem (10). We refer to a discretization satisfying (i) and (ii) as *conformal*, meaning conforming with the job parameters. To derive the modifications, we first re-interpret the variables:

- $y_{ai}$ : binary start indicator, i.e.,  $y_{ai} = 1 \iff t_{i-1} = t_a^*$  ( $a \in A_1$ ,  $i \in \mathcal{S}_a$ ), and
- $z_{ai}$ : binary indicator that job  $a$  is processed for all of interval  $i$  ( $a \in A_1$ ,  $i \in \mathcal{T}_a$ ).

Note that for a conformal discretization, any job starting at the start of a time interval must end at the end of a time interval. We now re-define

$$P_{ai} = \{k \in \mathcal{S}_a : t_{k-1} + p_a \geq t_i\},$$

which for a conformal discretization yields the set of intervals  $k$  for which if the job starts at  $t_{k-1}$  it is processed for the whole duration of interval  $i$ . Also observe that in this case, for  $k \in P_{ai}$ ,  $\mu_{aki}^+ = t_i - t_{i-1}$  and we re-define  $\mu_{aki}^- = t_i - t_{i-1}$  also. As a consequence, (30) simplifies to

$$z_{ji} = \sum_{k \in P_{ai}} y_{jk} \quad \forall a \in A_1, i \in \mathcal{T}_a. \quad (34)$$

The complete model is given by

$$\underline{\varphi} = \max_{(\mathbf{x}, \mathbf{y}, \mathbf{z}) \in F} \left\{ \sum_{i=1}^n \left( \sum_{a \in \delta^{\text{out}}(s)} x_{ai} - \sum_{a \in \delta^{\text{in}}(s)} x_{ai} \right) : \begin{array}{l} (2), (3), (4), (7), (8), \\ (26)-(29), (31), (32), (34), z \in \{0, 1\}^{\sum_{j \in J} |\mathcal{T}_j|} \end{array} \right\} \quad (35)$$

We have the following proposition.

**Proposition 4.** *Let  $0 = t_0 < t_1 < \dots < t_n = T$  be a conformal time discretization. Then  $\varphi^* \geq \underline{\varphi}$ , where  $\varphi^*$  is the optimal value of the original problem (10), and  $\underline{\varphi}$  is the optimal value of (35).*

As a consequence, if the discretization  $\Gamma$  contains  $\{t_a^* : a \in A_1\}$ , where  $\mathbf{t}^*$  is an optimal solution of (10), it must be that  $\underline{\varphi} = \text{val}(\mathbf{t}^*)$ .

Note that for integer data, the unit time discretization is conformal. Thus TDIP(TI) with the additional restriction  $z_{ai} \in \{0, 1\}$  for all  $a \in A_1$ ,  $i \in \mathcal{T}_a$ , yields a lower bound (and feasible solution) for (10). This model is also attractive since by Proposition 1 it yields an optimal solution for (10) in the case that there are no storage nodes. We denote this final variant with binary  $z$  variables by TDIP(TI)<sup>B</sup>.

The magnitude of  $n$  in any conformal discretization is likely to be large, and hence the lower bound integer programming model is likely to be slow to solve. Thus we also seek alternative methods of generating feasible solutions (and hence lower bounds) efficiently. For each of the formulations presented so far, the values of the  $z$  variables obtained by solving LP relaxations, for instance in a Branch&Bound tree, may contain information that can be useful in guiding construction heuristics for finding good schedules. Although such  $z$  values may be fractional, and may have positive values that, for a given job, are not consecutive over time, their “spread” across intervals and “intensity” within intervals may nevertheless be a useful guide for simple “repair” heuristics. We make these ideas precise in what follows, describing two such heuristics. Both

heuristics take as input a vector  $z$  with  $z_{ai} \in [0, 1]$  for all  $a \in A_1$  and  $i \in \mathcal{T}_a$ . Both derive start times  $t_a^*$  for all  $a \in A_1$ , which can then be evaluated by solving the max flow problem (1)–(9), so providing a lower bound for (10).

In our computational results, we apply these heuristics by solving a TDIP formulation with a given (relatively short) time limit; we then use the  $z$  vector from the LP solution at the node of the branch and bound tree yielding the best upper bound, and the  $z$  vector from the best feasible solution for the formulation found so far, as input to the heuristics, in turn.

## 5.1 The Projection heuristic

In this heuristic we use the “intensity” of the  $(z_a)_{i=1,\dots,n}$  variables for each  $a \in A_1$  as a guide to the choice of start time for the job,  $t_a^*$ . A given start time  $t_a^*$  can be thought of as inducing a vector  $(\xi_a)_{i=1,\dots,n}$  by interpreting  $\xi_{ai}$  as the proportion of interval  $i$  in which the job is being processed if the job starts at time  $t_a^*$  and is processed continuously until time  $t_a^* + p_a$ . This heuristic chooses the vector of start times  $t^*$  so as to minimize the  $\ell_1$ -norm distance between the  $\xi$  values induced by  $t^*$  and the given  $z$ . In this sense, the heuristic “projects”  $z$  onto the set of feasible schedules. Details of the method are as follows.

For  $t \in [r_a, d_a - p_a]$  and  $i \in \mathcal{T}_a$ , let  $l_{ai}(t)$  be the time for which job  $a$  is processed in interval  $i$  when it starts at time  $t$ , i.e.,

$$l_{ai}(t) = \begin{cases} \min\{t_i, t + p_a\} - \max\{t, t_{i-1}\} & \text{if } t < t_i \text{ and } t + p_a > t_{i-1}, \\ 0 & \text{otherwise.} \end{cases}$$

The total deviation between the actual processing times in the intervals and the given values  $(z_a)_{i=1,\dots,n}$  is measured by  $f(t) = \sum_{i \in \mathcal{T}_a} |(t_i - t_{i-1})z_{ai} - l_{ai}(t)|$ , and we choose the start time  $t_a^*$  as

$$t_a^* \in \arg \min \{f(t) : t \in [r_a, d_a - p_a]\}. \quad (36)$$

The value  $t_a^*$  can be determined as follows. For  $i \in \mathcal{S}_a$ , let  $\mathcal{E}_{ai} = \{k : t_{i-1} + p_a \leq t_k, t_i + p_a > t_{k-1}\}$  be the set of intervals in which job  $a$  can be completed when it starts in interval  $i$ . For  $k \in \mathcal{E}_{ai}$ , a minimizer of  $f(t)$  such that job  $a$  starts in interval  $i$  and ends in interval  $k$  is  $t_{aik}^* = t_i - \alpha$ , where  $(\alpha, \beta)$  is a minimizer of  $|\alpha - (t_i - t_{i-1})z_{ai}| + |\beta - (t_k - t_{k-1})z_{ak}|$  subject to  $\alpha + \beta = p_a - (t_{k-1} - t_i)$  and  $\alpha, \beta \geq 0$ , and finally we can put  $t_a^* = \arg \min \{f(t_{aik}^*) : i \in \mathcal{S}_a, k \in \mathcal{E}_{ai}\}$ .

## 5.2 The Centre-of-Mass heuristic

In this heuristic we use the “spread” of the  $(z_a)_{i=1,\dots,n}$  variables for each  $a \in A_1$  as a guide to the choice of start time for the job,  $t_a^*$ . For each arc  $a \in A_1$  we view the shutdown times  $\{(t_i - t_{i-1})z_{ai} : i \in \mathcal{T}_a\}$  as a distribution of the “mass” of its job processing time over the time horizon  $T$ . For  $t$  the unique point where this distribution of mass is balanced, (half is distributed earlier than  $t$  and half later), we schedule the job for arc  $a$  at time  $t_a^* = t - p_a/2$ . Details of the method are as follows.

Consider an arc in  $A_1$ , let  $i = \min\{i' : i' \in \mathcal{T}_a\}$  be the first interval that can be affected by job  $a$ , and let

$$h = \min \left\{ i' \in \mathcal{T}_a : \sum_{k=i}^{i'} (t_k - t_{k-1})z_{ak} > \frac{p_a}{2} \right\}.$$

We define the *midpoint* of job  $a$  (with respect to  $z$ ) to be the point  $t_{\text{mid}} \in [t_{h-1}, t_h]$  such that

$$\sum_{k=i}^{h-1} (t_k - t_{k-1})z_{ak} + (t_{\text{mid}} - t_{h-1})z_{ah} = \frac{p_a}{2},$$

and then we determine the start time  $t_a^*$  such that this midpoint exactly halves the processing period of job  $a$ , i.e.,  $t_a^* = t_{\text{mid}} - p_a/2$ .

## 6 Computational experiments

In this section we computationally evaluate, on a large set of instances, the performance of the exact formulation CTIP, the upper bounding formulations TDIP(RD) and TDIP(TI), and the lower bounding formulation TDIP(TI)<sup>B</sup>, together with the two LP based heuristics: the Centre of Mass (CoM) heuristic and the Projection Heuristic (Proj). We first describe the test data set, followed by the description of the performance measures used, before presenting the computational results.

### 6.1 Test instances

For our computational study, we use a subset of the randomly generated test instances from [5]. For each network, we consider the ten instances where all jobs have a time window in the range  $[25, 35]$  (the second, harder, instance set in [5]). These instances all have a time horizon of  $T = 1000$ . Some instance parameters and upper bound MIP dimensions are given in Table 1. We introduce storage in these instances by selecting one “central” node in each network to be a storage node. We tested a wide range of storage node capacities for that node. These tests showed that, for smaller networks, for values higher than about  $c_2 = 20$ , ( $c_2$  is a key parameter for arc capacities in the random generator), we found that the instances became very easy: in relatively short computing time (within a couple of minutes) the TDIP(TI)<sup>B</sup> model could be solved to give a feasible solution with the same value as the TDIP(RD) LP relaxation. Here we report results for all instances with storage capacities of 5, 10, 15 and 20, making a total of  $8 \times 10 \times 4 = 320$  instances tested.

We also investigate the performance of the formulations and heuristics on the two instances derived from a problem arising in the scheduling of maintenance for a coal supply chain, the Hunter Valley Coal Chain (HVCC), studied in [5]. The HVCC is the world’s largest coal export operation, handling coal mined in the Hunter Valley region of New South Wales, Australia. A record 150.5 million tonnes of coal was exported by the HVCC to customers around the world in the year 2013, with more than 1400 coal vessels per year served by the Port of Newcastle, located at the mouth of the Hunter River. There are three coal handling terminals at the port, for the storage of coal prior to loading onto vessels. Coal from around 35 coal mines is transferred to the coal handling terminals via a network of rail tracks spread over 450 km, using approximately 22,000 train trips per year. Each terminal has its storage space divided into stockpads on which coal stockpiles are assembled, with four stockpads each in Terminals 1 and 2, and with Terminal 3 modelled as a single stockpad. Coal is modelled as flowing through a network representing the combination of the rail network and the network of handling and conveying equipment at each terminal, with each stockpad modelled as a node with storage. The storage capacities for five of the stockpads is approximately 150 kilotonnes and for the remaining four is approximately 700 kilotonnes.

The components of this supply chain, such as rail tracks, conveyor belts, and stacking, reclaiming and shiploading machinery, all require regular preventive maintenance. However each maintenance plan has some negative impact on the throughput of network, as a component becomes unavailable for use during maintenance. Thus to meet increasing demand for coal, it is crucial to find maintenance schedules that minimize the impact of maintenance on the throughput of the system. MIP-based heuristics that consider only a sparse subset of possible maintenance job start times are used in [4] to obtain practical schedules, tested on 2010 and 2011 HVCC annual maintenance schedules. Seeking to combat the challenge posed by the very large number of possible job start times, matheuristics are developed and tested on two instances of the problem we consider here, derived from the same 2010 and 2011 HVCC schedules ([5]). However these matheuristics exploit the decomposable structure of the network flow problem that occurs if storage is ignored: the methods in [5] cannot be applied to the problem with storage and the instances tested in [5] disallow storage at the stockpad nodes. As demonstrated in Section 2.1, the optimal schedule for an instance of the problem obtained by ignoring storage can be very far from the optimal schedule for the instance when storage is considered. Here we analyze the performance of the formulations and heuristics described in this paper on the two instances used in [5], but considering storage at the stockpads, with their capacities set to the values given above. The time horizon for both instances is  $T = 365 \times 24 = 8760$  (with a discretization of 1 hour for one year). The 2010 and 2011 instances respectively contain 1457 and 1234 jobs. Every job has a time window of two weeks and a processing time between an hour and several days. The instance and upper bound MIP dimensions for these two instances are given in Table 2.

Network	Dimensions				TDIP(RD) Size				TDIP(TI) Size			
	Nodes	Arcs	Jobs	$ \mathcal{D} $	# Rows	# Columns	# Nonzeros	# Binaries	# Rows	# Columns	# Nonzeros	# Binaries
1	12	32	303.2	456.3	25 943.4	22 871.5	155 200.2	5883.6	62 288.0	51 149.5	682 573.4	9241.9
2	16	44	421.0	568.7	43 740.3	39 859.6	287 528.7	11 297.9	87 449.3	71 838.5	951 618.6	12 824.5
3	18	57	542.4	658.8	63 451.8	60 600.3	444 662.4	18 709.0	112 601.1	94 005.9	1 239 069.5	16 528.1
4	27	90	847.5	812.2	115 272.9	117 067.3	849 284.5	42 105.8	176 571.2	148 011.7	1 941 644.0	26 001.1
5	36	123	1155.9	897.5	214 480.0	180 212.0	2 172 605.0	31 506.4	239 884.6	201 350.0	2 647 223.0	35 258.8
6	32	92	873.8	818.3	149 522.0	122 561.0	1 389 731.0	21 681.1	183 799.3	150 383.2	1 993 952.6	26 643.1
7	48	176	1657.0	963.7	327 382.0	275 677.0	3 524 593.0	48 517.6	340 652.3	286 671.7	3 782 451.1	50 491.7
8	64	240	2268.2	987.5	459 361.0	387 151.0	5 064 772.0	68 177.5	465 869.9	392 573.7	5 194 776.4	69 172.4

Table 1: Sizes of the random networks and average(arithmetic mean) problem sizes for TDIP(RD) and TDIP(TI) after presolve. Each row summarizes statistics for ten instances, each with identical network and arc capacities, but different, randomly generated, jobs.

Year	Dimensions				TDIP(RD) Size				TDIP(TI) Size			
	Nodes	Arcs	Jobs	$ \mathcal{D} $	# Rows	# Columns	# Nonzeros	# Binaries	# Rows	# Columns	# Nonzeros	# Binaries
2010	109	176	1458	1990	293208	278451	2040868	54105	1125678	1554161	10429886	502577
2011	109	176	1235	1709	251706	235847	1653790	46284	1108575	1521901	10099289	493102

Table 2: Sizes of the HVCCC network and average problem sizes for TDIP(RD) and TDIP(TI) after presolve.

## 6.2 Performance Measures

To evaluate the quality of an upper bound for an instance  $I$ , we use the percentage gap between the upper bound and the best known lower bound for  $I$ , i.e.  $(\varphi_I - \varsigma_I^*)/\varsigma_I^* \times 100$  as a performance measure, where  $\varphi_I$  is the value of the upper bound and  $\varsigma_I^*$  is the value of the best lower bound for  $I$ .

Similarly, to evaluate the quality of a lower bound for an instance  $I$ , we use the percentage gap between the best known upper bound for  $I$  and the lower bound i.e.  $(\varphi_I^* - \varsigma_I)/\varsigma_I \times 100$  as a performance measure, where  $\varphi_I^*$  is the value of the best known upper bound for  $I$  and  $\varsigma_I$  is the value of the lower bound for  $I$ .

In order to compare the quality of different upper (lower) bounds we use performance profiles, in which, for each upper (lower) bound and a value  $g$  on the horizontal axis, we plot the percentage of instances that have the percentage gap to the best lower (upper) bound less than or equal to  $g\%$ .

## 6.3 Experimental Framework

The MILP formulations and heuristics are implemented in C++ and run on a Dell PowerEdge R710 with dual hex core 3.06GHz Intel Xeon X5675 processors and 96GB RAM running Red Hat Enterprise Linux 6. IBM ILOG CPLEX v12.5 is used in deterministic mode with a single thread. For each formulation we investigated the performance of different root algorithms, i.e. algorithms to solve the linear programming relaxation of the problem, provided in CPLEX. The primal simplex method performed the best overall and hence was used as the root algorithm for each formulation. For all formulations a time limit of 120 minutes (7200 seconds) to solve each randomly generated instance was imposed. For the much larger HVCC instances, we allowed twice as much time: the time limit was set to 240 minutes (14400 seconds). Each formulation was given an initial feasible solution in which a job  $j$  starts at time given by  $\lfloor (r_j + d_j - p_j)/2 \rfloor$ . All other CPLEX parameters were set to their default values.

In our computational study, we compare the quality of upper bounds given by CTIP, TDIP(RD) and TDIP(TI) formulations. We extract two upper bounds from each: (i) the LP relaxation value, denoted by LP-CTIP, LP-TDIP(RD) and LP-TDIP(TI) respectively, and (ii) the best bound given by the model at the end of the time limit, UB-CTIP, UB-TDIP(RD) and UB-TDIP(TI).

We also compare the quality of the lower bounds provided by CTIP and TDIP(TI)<sup>B</sup> at the end of the time limit, denoted by LB-CTIP and LB1 respectively. In addition to these, we also consider the following lower bounds for the computational analysis.

- CoM (Proj): the value of the feasible solution obtained by applying the CoM (Proj) heuristic to the

LP-relaxation solution of TDIP(RD)

- CoM-LP $\tau$  (Proj-LP $\tau$ ): the value of the feasible solution obtained by applying the CoM (Proj) heuristic to the LP relaxation solution at the active node with best upper bound found after solving TDIP(RD) for  $\tau$  seconds.
- CoM-FS $\tau$  (Proj-FS $\tau$ ): the value of the feasible solution obtained by applying the CoM (Proj) heuristic to the best feasible solution found by TDIP(RD) within  $\tau$  seconds.
- Max of All: the best feasible solution generated by any of the above heuristics.

For the randomly generated instances we use  $\tau = 300$ , but for the much larger HVCC instances, we use  $\tau = 1800$ .

We note here that throughout the next section all averages are taken to be geometric means (unless otherwise stated). Since percentage gaps in bounds can be zero, we use the *shifted* geometric mean with a shift of 1: the shifted geometric mean of values  $x_1, x_2, \dots, x_n$  with shift  $s$  is defined as  $(\prod_{i=1}^n (x_i + s))^{1/n} - s$ .

## 6.4 Results

We begin by comparing the quality of bounds given by CTIP and TDIP formulation on a modified smaller subset of the test data set, in which, for each instance in the original data set with storage capacity 5 (ten instances for each network giving a total of 80 instances), we discard all jobs with deadline greater than 60 and take the time horizon to be  $T = 60$ . The average number of jobs over the resulting ten instances for each network are shown in the second column of Table 3. The quality of bounds produced on these modified instances can be compared by observing Figures 7 and 8, with summary statistics given in Table 3. Run time statistics are given in Table 4. We first explain what is shown in each figure or table, and then summarize our findings from these results.

Figure 7 gives a performance profile for the percentage gap of each of the CTIP and TDIP model lower bounds (LB-CTIP and LB1 respectively). It also gives a dot plot, with two columns of dots for each network, one column for each for the CTIP and TDIP lower bounds. There are ten dots in each column, each plotted with  $y$ -axis value given by the percentage gap of the lower bound produced by that column's model on the corresponding instance. Each column also includes a box, plotted at a  $y$ -axis value given by the (shifted) geometric mean of the percentage gap produced by the model for that column over the ten instances represented. Figure 8 gives two plots with three performance profiles in each. The first provides the percentage gaps of each of the LP relaxation values of the CTIP, TDIP(RD) and TDIP(TI) formulations; the second provides the percentage gaps that were found by solving each of the CTIP, TDIP(RD) and TDIP(TI) MIPs with the given run time limit. Figure 8 includes a dot plot similar to that in Figure 7, but with three columns per network rather than two, one for each of the upper bounds UB-CTIP, UB-TDIP(RD) and UB-TDIP(TI).

Table 3 provides summary statistics for the performance of the CTIP and TDIP model bounds. Its third and eighth columns report the number of instances, out of ten, for which best feasible solution found by the CTIP and TDIP(TI)<sup>B</sup> MIPs within the run time limit is known to be optimal (has value equal to the best upper bound found for the instance with any model, i.e. has zero percentage gap), respectively. The table's fourth and fifth column report information "internal" to the CTIP MIP solution process, showing the number of instances (out of ten) for which the CTIP model was able to prove optimality of its feasible solution within the time limit, and the average (across the ten instances) of the gap between its best upper and best lower bounds at the end of the time limit, reported as a percentage. The table's sixth column reports the average percentage gap of the CTIP lower bound (LB-CTIP, calculated using the best upper bound found by any model). Similarly the seventh column reports the average percentage gap of the CTIP upper bound (UB-CTIP). The ninth column shows the average percentage gap of the lower bound produced by the TDIP model (i.e., LB1, from TDIP(TI)<sup>B</sup>). The tenth and eleventh columns respectively report the average percentage gap of the two TDIP upper bounds, UB-TDIP(RD) and UB-TDIP(TI). Recall that all averages are shifted geometric means and that percentage gaps are all calculated with respect to the best bound produced by any model, with the exception of those in the fifth column, which are as explained above.



Summary statistics for the run times of CTIP and TDIP models on the modified dataset are given in Table 4. For each model, the minimum, maximum and average (geometric mean) run time, in seconds, across the ten instances for each network, are shown.

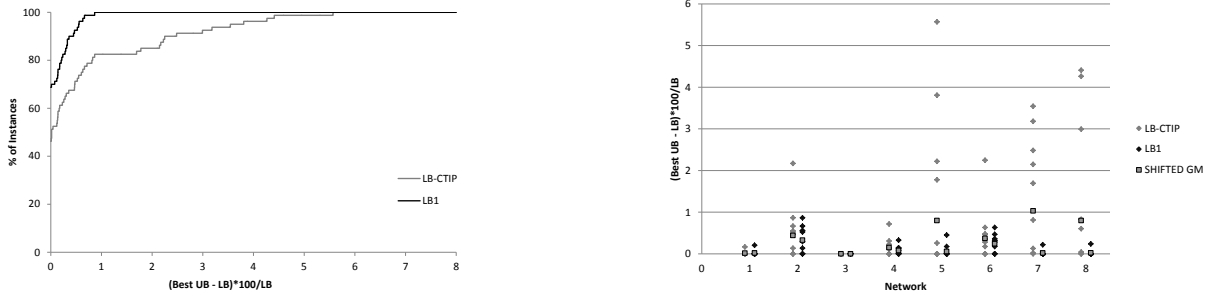


Figure 7: Comparison of lower bounds on the modified data set.

Our first observation from these results is that the CTIP MIP model struggles to solve to optimality. Even on these very small modified instances, it times out on 44 of the 80 instances. By comparison, on these instances, almost none of the TDIP formulations reached the run time limit: the only exception was the TDIP(TI) formulation, which timed out on 2 of the 20 instances for Networks 7 and 8.

In terms of the quality of lower bounds produced, the CTIP model was clearly outperformed by the TDIP(TI)<sup>B</sup> model. This can be immediately observed by comparing their performance profiles in Figure 7, in which the profile for LB1 remains well above and to the left of that for LB-CTIP, until they converge at a percentage gap of around 6%. For all instances LB1 is within 2% of the best upper bound whereas LB-CTIP is within 2% for only 80% of instances. In the dot plot, we see the second column (for LB1) is typically much shorter, with the first column (for LB-CTIP) showing a spread with several instances' dots plotted higher than the height of the second column. From Table 3, we see that only 46.25% instances are solved to optimality by CTIP, i.e. LB-CTIP is optimal, versus 68.75% for LB1. For every network the average percentage gap of LB1 is smaller than that of LB-CTIP, with the relative difference between the gaps increasing with increasing size of the instances. From Table 4 we see that run times for TDIP(TI)<sup>B</sup> are also usually substantially shorter (in some cases by several orders of magnitude) than those for CTIP.

In regard to upper bounds, the TDIP models also offer a much better trade-off for quality versus run time when compared to the CTIP model. From the first performance profile plot in Figure 8, we see that the quality of upper bound from the CTIP model's LP relaxation is similar to that of TDIP(TI), but as can be seen from Table 4, the run times for the latter are generally lower, and more so for the larger instances. From the second performance profile plot in Figure 8, we see that the quality of upper bound from the CTIP MIP is similar to that of TDIP(RD) (and noticeably worse than that of TDIP(TI)), but as can be seen from Table 4, the run times for the TDIP(RD) (and TDIP(TI)) MIPs are generally lower, by several orders of magnitude in the case of TDIP(RD).

We conclude that the CTIP model is primarily of theoretical value, allowing us to provide insights about the nature of optimal solutions but struggling to solve problems in practice. From the experiments on instances that are much smaller than those in the original data set, it seems very unlikely that the CTIP model could perform well on realistic sized instances, and appears to be outperformed in all respects by the TDIP models. Thus in what follows do not report results for the CTIP model on the full sized and real world instances; we now focus on the performance of the TDIP models and the heuristics on these instances.

Figure 9 provides performance profiles for percentage gaps of the lower bounds, with two plots: one for the instances with Networks 1-4, which are markedly easier, and the other for the instances with Networks 5-8. Summary statistics for the performance of the lower bounds on the randomly generated instances are given in Table 5, which shows the minimum, maximum and average (shifted geometric mean) percentage gaps of each heuristic, over all ten instances with each network and each storage capacity level. The same statistics taken over all forty instances with each network are reported in bold font. Table 6 is similar, but

Network		CTIP						TDIP(TI) <sup>B</sup>			TDIP(RD)			TDIP(TI)		
No.	Avg. # Jobs	# Opt	# CTIP Opt	% gap & LB-CTIP	% gap UB-CTIP & LB-CTIP	%gap Best UB & LB-CTIP	% gap UB-CTIP & Best LB	# Opt	% gap Best UB & LB1	UB	% gap UB-TDIP (RD) & Best LB	UB-TDIP (RD) & Best LB	UB-TDIP (TI) & Best LB	% gap UB-TDIP (TI) & Best LB	UB-TDIP (TI) & Best LB	UB-TDIP (TI) & Best LB
1	7.93	9	9	0.06	0.06	0.02	0.06	9	0.02		0.16			0.04		
2	11.89	3	3	1.68	1.68	0.44	1.54	3	0.33		1.66			0.37		
3	12.08	10	10	0.00	0.00	0.00	0.00	10	0.00		0.00			0.00		
4	20.93	4	4	0.63	0.63	0.15	0.52	4	0.08		0.37			0.07		
5	28.79	5	5	0.90	0.90	0.80	0.19	8	0.05		0.10			0.05		
6	20.71	3	2	1.62	1.62	0.37	1.43	3	0.24		1.43			0.26		
7	38.48	1	1	1.05	1.05	1.03	0.05	9	0.02		0.02			0.02		
8	56.14	2	2	0.80	0.80	0.80	0.02	9	0.02		0.02			0.02		

Table 3: Comparison of CTIP with TDIP on the modified data set.

Network		CTIP						TDIP											
No.		LP-CTIP			CTIP			TDIP(RD)				LP-TDIP(TI)				TDIP(TI)			
		min	avg	max	min	avg	max	min	avg	max	min	min	avg	max	min	min	avg	max	max
1		0.01	0.01	0.02	0.03	3.45	7200	0.01	0.01	0.02	0.01	0.01	0.02	0.03	0.06	0.35	33.51	0.06	3.68
2		0.03	0.21	0.73	0.11	1061.01	7200	0.01	0.03	0.05	0.02	0.02	0.05	0.10	0.07	3.46	88.44	0.14	11.78
3		0.03	0.06	0.20	0.07	3.40	112.92	0.01	0.02	0.03	0.02	0.02	0.03	0.06	0.09	0.13	0.28	0.09	0.23
4		0.11	0.19	0.83	51.69	2078.91	7200	0.03	0.06	0.23	0.05	0.08	0.08	0.28	0.17	4.26	1979.58	0.19	1136.56
5		0.33	0.63	3.46	0.59	1937.75	7200	0.04	0.08	0.24	0.08	0.13	0.13	0.23	0.36	4.92	688.24	0.37	132.61
6		0.18	0.41	1.90	104.74	4319.72	7200	0.03	0.10	0.41	0.03	0.11	0.11	0.24	0.15	17.12	1798.90	0.19	70.86
7		0.29	0.94	4.09	1.97	3171.03	7200	0.09	0.13	0.31	0.10	0.15	0.15	0.45	0.34	2.79	7200	0.38	448.22
8		0.92	2.25	5.98	2.56	1472.38	7200	0.15	0.28	1.29	0.14	0.25	0.25	0.59	0.45	2.64	7200	0.55	759.88

Table 4: Run time statistics of MIPs on the modified data set, reported in seconds.

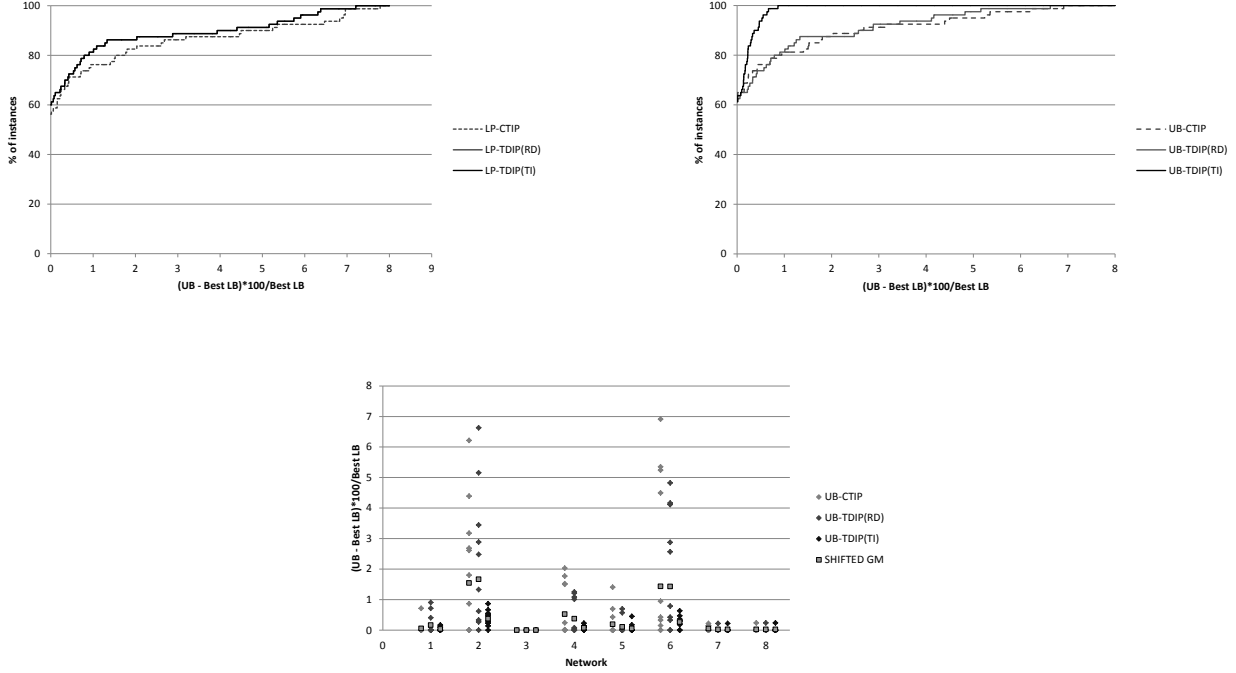


Figure 8: Comparison of upper bounds on the modified data set.

shows statistics for the performance of the upper bounds. Percentage gaps of lower and upper bounds on the HVCC instances are shown Tables 9 and 8 respectively. Run time statistics are reported in Table 7 for the randomly generated instances and in Table 10 for the HVCC instances. In the former case, the columns labelled “# time limit” record the number of instances on which the corresponding model timed out, out of the number represented by the row statistics (ten or forty).

First, we observe that all TDIP formulations struggled to solve these instances to optimality. None of the TDIP(RD), TDIP(TI) or TDIP(TI)<sup>B</sup> formulations solved the HVCC instances to optimality within the time limit. As can be seen from Table 7, the formulations all reached the time limit for almost all randomly generated instances with networks other than Networks 1 and 3. For Network 3 all instances solved to optimality with all formulations, while for Network 1, TDIP(RD) solved to optimality for all instances, but TDIP(TI) only solved 17, while TDIP(TI)<sup>B</sup> could only solve 14. The TDIP(RD) formulation solved more instances to optimality than any other, but this was still only 91 instances out of 320. We note that instances seem to get more difficult as the storage capacity decreases, as well as with increasing network size and number of jobs.

Fortunately, even without solving to optimality, the TDIP(RD) and TDIP(TI) formulations provide quite good upper bounds. As can be seen from Table 6, UB-TDIP(TI) appears to provide the best upper bounds overall on randomly generated instances, with average percentage gaps less than 2% for all networks, and less than 2.3% for all networks and storage capacity levels, except for those with Network 6, which has quite large gaps, averaging 6.87%. For Network 6, UB-TDIP(RD) appears noticeably better, and is very close to UB-TDIP(TI) on the other networks. For the HVCC instances reported in Table 10, UB-TDIP(TI) is best for the 2010 instance, but UB-TDIP(RD) is best for 2011. For most randomly generated instances, the LP relaxations actually provide upper bounds that are nearly as good as those given by the MIPs, and of course take much less time (as per Table 7). The exception is Network 2, for which there is a noticeable improvement in the quality of the upper bound resulting from solving the MIP rather than just its LP relaxation. Noticeable improvements are also observed for the HVCC instances, particularly the 2010 instance. From these results it is difficult to “pick a winner” between UB-TDIP(TI) and UB-TDIP(RD), but it does appear that UB-TDIP(RD) does relatively well in cases where  $|\mathcal{D}|$  is smaller: the 2011 HVCC

instance has  $|\mathcal{D}|$  value quite a bit smaller than that for 2010, and Network 6 has the smallest  $|\mathcal{D}|$  of all of the harder networks (5-8). One possibility is that when there are fewer unique job start and end times, the more compact TDIP(RD) formulation offers a better trade-off between MIP solvability and bound quality than does TDIP(TI).

The lower bounds show quite a bit more diversity than the upper bounds in terms of offering a trade-off of quality versus run time. It is clear that with long run times, LB1 (found by solving the MIP TDIP(TI)<sup>B</sup>) is by a large margin the best lower bound on randomly generated instances: it gives feasible solutions that can be proved to be within 2.06% of optimality for all instances with Networks 1-4 and gives feasible solutions within 5% of optimality on more than 80% of instances with Networks 5-8 (see Table 5 and Figure 9). No other lower bound is better than LB1 on any randomly generated instance. The situation on HVCC instances (Table 9) is somewhat different: although LB1 is very close to the best on the 2010 instance, it is very far from best on the 2011 instance. Here the CoM and Proj heuristic methods come into their own.

The performance of all variants of the CoM and Proj heuristics, as described in Section 6.3, on the smaller randomly generated instances (those with Networks 1-4) are shown in Figure 9. From this plot we observe that the methods based on the initial LP relaxation solution of TDIP(RD) perform very poorly: although they are very fast to run, with only a small amount of extra computing time (5 minutes), very substantial improvements in the bound quality can be made. The same conclusion can be reached by comparing the columns for “Max(CoM,Proj)” with those for “Max(Proj-LP300,Proj-FS300)” and “Max(CoM-LP,CoM-FS300)” in Table 5; the latter show far smaller percentage gaps. Thus in the plot for Networks 5-8 in Figure 9) we focus on the heuristics applied to either best upper bound LP solutions or best lower bound integer feasible solutions found after 5 minutes of computing time. Here it can be seen that the LP solutions seem to be a better basis for the heuristics than the integer feasible solutions, with both Proj-LP300 and CoM-LP300 profiles lying above the Proj-FS300 and CoM-FS300 profiles, with some margin. This can be explained by the fact that for these instances the first LP relaxation takes quite a long time to solve, (around 200 seconds on average), hence for most instances the MIP solver could not improve on the initial feasible solution provided for TDIP(RD) within the 300 second limit. We also note that the Proj heuristic seems slightly better than CoM: the difference is small when based on the integer feasible solution, but is noticeable when the heuristics are based on the LP solution. This can also be observed by comparing the columns for “Max(Proj-LP300,Proj-FS300)” with those of “Max(CoM-LP,CoM-FS300)” in Table 5. The situation on HVCC instances is quite different. Here the heuristics based on the best integer feasible solutions at the (longer) 1800 second mark are actually better than those based on the LP solution giving the best upper bound at that time, by quite a large margin. Although the time limit at which to extract the lower bound relative to the size or difficulty of the instance is difficult to calibrate, this would suggest that the integer feasible solution offers an increasingly better basis for the heuristics as longer TDIP(RD) MIP run times are allowed.

Since the CoM and Proj heuristics are computationally very cheap to run, and one variant is not consistently better than another on all instances, we also consider running all variants after the TDIP(RD) MIP lower bound time limit is reached (“Max of All”). As can be seen from the performance profiles on the harder randomly generated instances (Networks 5-8) in Figure 9, and by comparing the columns for “Max of All” in Table 5 with those for the other heuristic variants, this gives noticeable improvements over the results of any one (or pair of) heuristics alone, at virtually no extra computational cost. Whilst the quality of the lower bound produced by “Max of All” is still nowhere near as good as LB1 on randomly generated instances, it comes at far less computational cost, and has the added advantage of “controllability” of the computational cost via the time at which the solutions on which to base the heuristics are extracted. Indeed, it gives the best quality lower bounds on the HVCC instances when these solutions are extracted at 1800 seconds.

Network	Storage Capacity	%gap Best UB & LB1			%gap Best UB & Max(CoM,Proj)			%gap Best UB & Max(CoM-LP300, CoM-FS300)			%gap Best UB & Max(Proj-LP300, Proj-FS300)			%gap Best UB & Max of All		
		min	avg	max	min	avg	max	min	avg	max	min	avg	max	min	avg	max
1	5	0.00	0.08	0.27	5.93	11.44	16.28	1.64	3.15	5.71	1.61	3.14	5.90	1.61	3.06	5.71
	10	0.00	0.04	0.23	6.48	12.14	17.41	1.56	2.97	5.25	1.62	3.13	5.33	1.56	2.96	5.25
	15	0.00	0.02	0.11	5.24	11.63	15.47	1.68	3.16	5.07	1.54	3.22	5.39	1.54	3.12	5.07
	20	0.00	0.01	0.03	7.00	10.76	14.55	1.22	2.90	5.55	1.33	2.96	5.71	1.22	2.89	5.55
	<b>Average</b>	<b>0.00</b>	<b>0.04</b>	<b>0.27</b>	<b>5.24</b>	<b>11.48</b>	<b>17.41</b>	<b>1.22</b>	<b>3.04</b>	<b>5.71</b>	<b>1.33</b>	<b>3.11</b>	<b>5.90</b>	<b>1.22</b>	<b>3.00</b>	<b>5.71</b>
2	5	0.22	1.37	2.06	14.72	18.83	25.34	5.53	7.84	12.73	4.08	7.52	12.82	4.08	7.37	12.73
	10	0.22	1.31	1.93	11.92	18.92	28.53	4.36	7.10	11.31	5.29	7.47	10.30	4.36	7.02	10.30
	15	0.16	1.22	1.87	13.34	18.45	25.89	4.01	6.90	10.34	3.92	7.23	10.96	3.92	6.86	10.29
	20	0.12	1.08	1.80	12.39	18.30	27.49	3.80	6.57	10.24	2.89	6.69	10.21	2.89	6.37	10.21
	<b>Average</b>	<b>0.12</b>	<b>1.24</b>	<b>2.06</b>	<b>11.92</b>	<b>18.62</b>	<b>28.53</b>	<b>3.80</b>	<b>7.09</b>	<b>12.73</b>	<b>2.89</b>	<b>7.22</b>	<b>12.82</b>	<b>2.89</b>	<b>6.89</b>	<b>12.73</b>
3	5	0.00	0.00	0.00	8.01	11.14	15.87	0.57	1.58	2.82	0.58	1.61	2.79	0.57	1.57	2.79
	10	0.00	0.00	0.00	6.88	10.56	15.89	1.01	1.63	2.52	1.08	1.68	2.34	1.01	1.60	2.28
	15	0.00	0.00	0.00	6.58	10.34	14.28	0.78	1.56	2.28	0.83	1.60	2.04	0.78	1.53	2.04
	20	0.00	0.00	0.00	6.01	10.69	15.56	1.17	1.59	2.28	1.15	1.69	2.31	1.15	1.59	2.28
	<b>Average</b>	<b>0.00</b>	<b>0.00</b>	<b>0.00</b>	<b>6.01</b>	<b>10.68</b>	<b>15.89</b>	<b>0.57</b>	<b>1.59</b>	<b>2.82</b>	<b>0.58</b>	<b>1.64</b>	<b>2.79</b>	<b>0.57</b>	<b>1.57</b>	<b>2.79</b>
4	5	0.00	0.40	1.42	7.35	11.20	19.36	1.20	3.88	12.37	0.91	2.70	10.49	0.91	2.69	10.49
	10	0.00	0.32	1.26	6.36	10.94	20.05	1.01	3.73	14.51	1.10	3.71	8.63	1.01	3.35	8.63
	15	0.00	0.21	0.74	6.41	10.48	17.73	0.36	3.00	9.76	0.36	2.80	9.93	0.36	2.77	9.76
	20	0.00	0.18	0.54	6.42	9.93	18.43	0.76	2.81	9.02	1.02	3.22	11.99	0.76	2.62	9.02
	<b>Average</b>	<b>0.00</b>	<b>0.27</b>	<b>1.42</b>	<b>6.36</b>	<b>10.63</b>	<b>20.05</b>	<b>0.36</b>	<b>3.33</b>	<b>14.51</b>	<b>0.36</b>	<b>3.09</b>	<b>11.99</b>	<b>0.36</b>	<b>2.85</b>	<b>10.49</b>
5	5	0.18	2.24	6.84	9.46	13.65	18.22	5.98	12.47	21.11	5.39	11.75	19.73	5.39	10.94	18.22
	10	0.26	2.06	3.76	9.91	13.52	18.02	8.56	12.34	16.71	8.31	12.90	19.31	8.31	11.74	16.62
	15	0.10	1.77	3.83	10.16	13.76	17.96	6.33	12.39	17.33	5.93	12.23	17.99	5.93	11.72	17.02
	20	0.23	1.74	3.11	9.67	13.46	17.05	9.69	12.92	18.94	8.39	12.86	19.62	8.39	11.96	17.05
	<b>Average</b>	<b>0.10</b>	<b>1.94</b>	<b>6.84</b>	<b>9.46</b>	<b>13.60</b>	<b>18.22</b>	<b>5.98</b>	<b>12.53</b>	<b>21.11</b>	<b>5.39</b>	<b>12.42</b>	<b>19.73</b>	<b>5.39</b>	<b>11.58</b>	<b>18.22</b>
6	5	3.67	6.45	9.27	16.70	21.82	26.81	12.74	21.58	38.11	11.69	19.27	33.46	11.69	18.43	26.81
	10	2.91	6.62	10.76	15.75	21.97	27.13	12.93	20.81	33.03	11.47	18.90	29.90	11.47	18.17	27.13
	15	2.51	6.36	12.01	16.88	22.80	31.06	13.59	21.28	35.18	11.43	19.15	31.48	11.43	19.07	31.06
	20	4.29	6.93	11.54	18.52	23.66	30.10	13.26	19.88	30.44	11.92	19.65	28.56	11.92	18.81	28.56
	<b>Average</b>	<b>2.51</b>	<b>6.59</b>	<b>12.01</b>	<b>15.75</b>	<b>22.55</b>	<b>31.06</b>	<b>12.74</b>	<b>20.88</b>	<b>38.11</b>	<b>11.43</b>	<b>19.24</b>	<b>33.46</b>	<b>11.43</b>	<b>18.62</b>	<b>31.06</b>
7	5	0.06	0.70	2.23	4.84	9.53	12.27	4.30	8.33	12.64	3.54	7.38	12.35	3.54	7.09	11.45
	10	0.02	0.64	2.88	5.41	9.73	12.58	3.48	7.79	11.35	3.35	8.07	13.69	3.35	7.51	10.92
	15	0.00	0.65	3.11	5.20	9.71	14.02	1.89	6.79	10.72	2.16	6.44	11.56	1.89	6.08	9.16
	20	0.00	0.49	1.84	4.81	9.26	12.60	4.32	8.21	12.13	3.74	7.16	10.43	3.74	6.98	10.43
	<b>Average</b>	<b>0.00</b>	<b>0.62</b>	<b>3.11</b>	<b>4.81</b>	<b>9.55</b>	<b>14.02</b>	<b>1.89</b>	<b>7.76</b>	<b>12.64</b>	<b>2.16</b>	<b>7.24</b>	<b>13.69</b>	<b>1.89</b>	<b>6.90</b>	<b>11.45</b>
8	5	0.01	1.10	3.83	5.78	10.68	16.93	4.70	10.35	17.95	4.64	9.38	17.97	4.64	9.00	16.93
	10	0.01	1.15	5.44	6.35	10.58	19.82	2.90	9.48	16.39	3.46	9.02	16.39	2.90	8.46	16.39
	15	0.00	1.50	6.77	5.72	10.64	17.46	4.94	9.45	17.73	4.28	9.10	17.87	4.28	8.85	17.46
	20	0.00	1.20	7.11	5.42	10.52	17.75	4.07	9.26	19.11	5.49	10.10	16.81	4.07	8.73	16.81
	<b>Average</b>	<b>0.00</b>	<b>1.23</b>	<b>7.11</b>	<b>5.42</b>	<b>10.60</b>	<b>19.82</b>	<b>2.90</b>	<b>9.63</b>	<b>19.11</b>	<b>3.46</b>	<b>9.39</b>	<b>17.97</b>	<b>2.90</b>	<b>8.76</b>	<b>17.46</b>

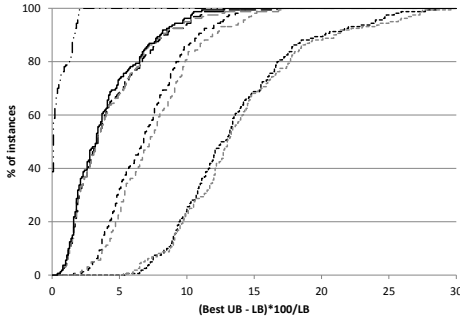
Table 5: Minimum, average (shifted geometric mean) and maximum relative percentage gaps between lower bounds and the best upper bound.

Network	Storage Capacity	%gap LP-TDIP(RD) & Best LB			%gap UB-TDIP(RD) & Best LB			%gap LP-TDIP(TI) & Best LB			%gap UB-TDIP(TI) & Best LB		
		min	avg	max	min	avg	max	min	avg	max	min	avg	max
1	5	0.00	0.15	0.36	0.00	0.10	0.27	0.00	0.14	0.36	0.00	0.09	0.36
	10	0.00	0.10	0.28	0.00	0.06	0.23	0.00	0.09	0.27	0.00	0.04	0.25
	15	0.00	0.07	0.24	0.00	0.04	0.15	0.00	0.06	0.23	0.00	0.02	0.11
	20	0.00	0.05	0.21	0.00	0.03	0.11	0.00	0.05	0.21	0.00	0.01	0.03
	<b>Average</b>	<b>0.00</b>	<b>0.09</b>	<b>0.36</b>	<b>0.00</b>	<b>0.06</b>	<b>0.27</b>	<b>0.00</b>	<b>0.09</b>	<b>0.36</b>	<b>0.00</b>	<b>0.04</b>	<b>0.36</b>
2	5	0.51	2.11	3.29	0.22	1.55	2.49	0.50	2.08	3.25	0.29	1.38	2.06
	10	0.51	1.98	3.12	0.22	1.47	2.37	0.49	1.95	3.10	0.28	1.32	1.93
	15	0.46	1.88	3.07	0.16	1.37	2.31	0.44	1.86	3.06	0.25	1.24	1.87
	20	0.42	1.73	2.97	0.12	1.25	2.34	0.40	1.71	2.97	0.22	1.10	1.80
	<b>Average</b>	<b>0.42</b>	<b>1.92</b>	<b>3.29</b>	<b>0.12</b>	<b>1.41</b>	<b>2.49</b>	<b>0.40</b>	<b>1.90</b>	<b>3.25</b>	<b>0.22</b>	<b>1.26</b>	<b>2.06</b>
3	5	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	15	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	20	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	<b>Average</b>	<b>0.00</b>	<b>0.00</b>	<b>0.00</b>	<b>0.00</b>	<b>0.00</b>	<b>0.00</b>	<b>0.00</b>	<b>0.00</b>	<b>0.00</b>	<b>0.00</b>	<b>0.00</b>	<b>0.00</b>
4	5	0.00	0.40	1.44	0.00	0.40	1.42	0.00	0.40	1.44	0.00	0.40	1.42
	10	0.00	0.32	1.27	0.00	0.32	1.26	0.00	0.32	1.27	0.00	0.32	1.26
	15	0.00	0.21	0.77	0.00	0.21	0.74	0.00	0.21	0.76	0.00	0.21	0.74
	20	0.00	0.18	0.56	0.00	0.18	0.54	0.00	0.18	0.55	0.00	0.18	0.54
	<b>Average</b>	<b>0.00</b>	<b>0.28</b>	<b>1.44</b>	<b>0.00</b>	<b>0.27</b>	<b>1.42</b>	<b>0.00</b>	<b>0.28</b>	<b>1.44</b>	<b>0.00</b>	<b>0.27</b>	<b>1.42</b>
5	5	0.18	2.24	6.86	0.18	2.24	6.84	0.18	2.24	6.85	0.18	2.24	6.84
	10	0.26	2.07	3.77	0.26	2.06	3.76	0.26	2.07	3.77	0.26	2.06	3.76
	15	0.10	1.77	3.84	0.10	1.77	3.84	0.10	1.77	3.84	0.10	1.77	3.83
	20	0.23	1.74	3.11	0.23	1.74	3.11	0.23	1.74	3.11	0.23	1.74	3.11
	<b>Average</b>	<b>0.10</b>	<b>1.95</b>	<b>6.86</b>	<b>0.10</b>	<b>1.94</b>	<b>6.84</b>	<b>0.10</b>	<b>1.95</b>	<b>6.85</b>	<b>0.10</b>	<b>1.94</b>	<b>6.84</b>
6	5	3.97	6.78	9.39	3.67	6.47	9.28	3.96	6.76	9.36	3.72	6.50	9.27
	10	3.14	6.89	10.85	2.91	6.63	10.76	3.11	6.87	10.83	2.92	6.69	10.76
	15	2.68	6.63	12.29	2.51	6.37	12.10	2.67	6.61	12.25	2.52	6.43	12.01
	20	4.53	7.20	11.76	4.29	6.94	11.55	4.53	7.18	11.72	4.40	7.02	11.54
	<b>Average</b>	<b>2.68</b>	<b>6.87</b>	<b>12.29</b>	<b>2.51</b>	<b>6.60</b>	<b>12.10</b>	<b>2.67</b>	<b>6.85</b>	<b>12.25</b>	<b>2.52</b>	<b>6.66</b>	<b>12.01</b>
7	5	0.06	0.70	2.23	0.06	0.70	2.23	0.06	0.70	2.23	0.06	0.70	2.23
	10	0.02	0.64	2.88	0.02	0.64	2.88	0.02	0.64	2.88	0.02	0.64	2.88
	15	0.00	0.65	3.11	0.00	0.65	3.11	0.00	0.65	3.11	0.00	0.65	3.11
	20	0.00	0.49	1.84	0.00	0.49	1.84	0.00	0.49	1.84	0.00	0.49	1.84
	<b>Average</b>	<b>0.00</b>	<b>0.62</b>	<b>3.11</b>	<b>0.00</b>	<b>0.62</b>	<b>3.11</b>	<b>0.00</b>	<b>0.62</b>	<b>3.11</b>	<b>0.00</b>	<b>0.62</b>	<b>3.11</b>
8	5	0.01	1.10	3.83	0.01	1.10	3.83	0.01	1.10	3.83	0.01	1.10	3.83
	10	0.01	1.15	5.44	0.01	1.15	5.44	0.01	1.15	5.44	0.01	1.15	5.44
	15	0.00	1.50	6.77	0.00	1.50	6.77	0.00	1.50	6.77	0.00	1.50	6.77
	20	0.00	1.20	7.11	0.00	1.20	7.11	0.00	1.20	7.11	0.00	1.20	7.11
	<b>Average</b>	<b>0.00</b>	<b>1.23</b>	<b>7.11</b>	<b>0.00</b>	<b>1.23</b>	<b>7.11</b>	<b>0.00</b>	<b>1.23</b>	<b>7.11</b>	<b>0.00</b>	<b>1.23</b>	<b>7.11</b>

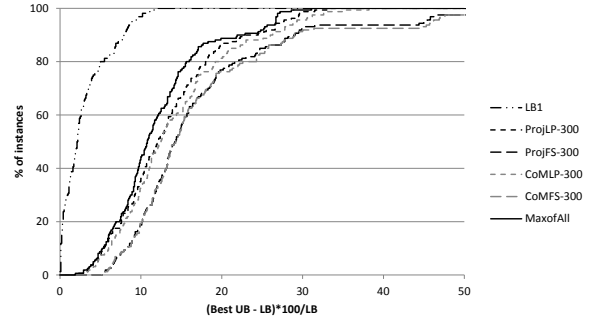
Table 6: Minimum, average (shifted geometric mean) and maximum percentage gaps between upper bounds and the best lower bound.

Network	Storage capacity	LP-TDIP(RD)				UB-TDIP(RD)				LP-TDIP(TI)				UB-TDIP(TI)				LBI			
		min	avg	max	# time lim	min	avg	max	# time lim	min	avg	max	# time lim	min	avg	max	# time lim	min	avg	max	# time limit
1	5	0.39	0.68	1.14	0	7.56	123.85	1932.99	0	1.33	1.78	2.29	0	1108.14	4794.57	7200	6	82.83	4029.62	7200	8
	10	0.43	0.61	0.76	0	20.46	74.87	248.73	0	1.38	1.90	2.70	0	68.37	2046.89	7200	6	139.16	1375.25	7200	2
	15	0.50	0.67	0.78	0	9.72	52.15	139.94	0	1.37	2.06	2.90	0	110.40	1661.98	7200	4	80.83	854.58	7200	4
	20	0.36	0.49	0.67	0	8.51	40.13	103.25	0	1.30	1.86	2.29	0	57.64	955.02	7200	1	32.11	224.784	1078.05	0
	Average	<b>0.36</b>	<b>0.60</b>	<b>1.14</b>	<b>0</b>	<b>7.56</b>	<b>66.37</b>	<b>1932.99</b>	<b>0</b>	<b>1.30</b>	<b>1.90</b>	<b>2.90</b>	<b>0</b>	<b>57.64</b>	<b>1986.65</b>	<b>7200</b>	<b>17</b>	<b>32.11</b>	<b>1015.76</b>	<b>7200</b>	<b>14</b>
2	5	2.45	4.29	8.50	0	7200	7200	7200	10	4.88	11.44	31.07	0	7200	7200	7200	10	7200	7200	7200	10
	10	1.82	3.77	7.29	0	7200	7200	7200	10	6.01	11.25	23.41	0	7200	7200	7200	10	7200	7200	7200	10
	15	2.25	4.18	6.95	0	7200	7200	7200	10	5.43	11.71	21.72	0	7200	7200	7200	10	7200	7200	7200	10
	20	1.52	2.95	5.18	0	7200	7200	7200	10	5.42	11.42	19.25	0	7200	7200	7200	10	7200	7200	7200	10
	Average	<b>1.52</b>	<b>3.76</b>	<b>8.50</b>	<b>0</b>	<b>7200</b>	<b>7200</b>	<b>7200</b>	<b>40</b>	<b>4.88</b>	<b>11.46</b>	<b>31.07</b>	<b>0</b>	<b>7200</b>	<b>7200</b>	<b>7200</b>	<b>40</b>	<b>7200</b>	<b>7200</b>	<b>7200</b>	<b>40</b>
3	5	1.64	1.89	2.17	0	19.67	46.16	115.63	0	2.04	2.37	2.67	0	48.19	245.75	431.93	0	46.94	92.138	128.93	0
	10	1.33	1.60	2.15	0	18.62	31.33	53.32	0	2.35	2.79	3.21	0	117.78	322.78	757.44	0	53.21	80.5027	150.93	0
	15	1.75	2.12	2.91	0	18.87	41.03	99.18	0	2.61	2.85	3.37	0	109.68	226.16	533.15	0	55.05	82.5626	117.37	0
	20	1.26	1.52	1.86	0	18.36	38.86	124.35	0	2.25	2.63	2.97	0	111.99	228.64	875.31	0	49.73	90.7722	162.99	0
	Average	<b>1.26</b>	<b>1.77</b>	<b>2.91</b>	<b>0</b>	<b>18.36</b>	<b>38.97</b>	<b>124.35</b>	<b>0</b>	<b>2.04</b>	<b>2.65</b>	<b>3.37</b>	<b>0</b>	<b>48.19</b>	<b>253.07</b>	<b>875.31</b>	<b>0</b>	<b>46.94</b>	<b>86.35</b>	<b>162.99</b>	<b>0</b>
4	5	6.02	15.31	36.42	0	3305.32	6660.71	7200	9	7.75	17.25	47.90	0	2659.50	6517.47	7200	9	7200	7200	7200	10
	10	8.10	16.17	42.79	0	1664.31	5705.44	7200	8	7.62	17.40	40.44	0	3477.27	6375.37	7200	8	1441.61	6130.33	7200	9
	15	6.52	17.73	41.12	0	418.42	5115.78	7200	6	7.76	17.13	40.41	0	5244.40	6905.67	7200	8	2007	6336.57	7200	9
	20	5.28	11.70	22.48	0	643.63	4525.00	7200	6	8.04	15.25	32.50	0	4296.82	6487.23	7200	7	1173.74	4926.81	7200	7
	Average	<b>5.28</b>	<b>15.05</b>	<b>42.79</b>	<b>0</b>	<b>418.42</b>	<b>5446.09</b>	<b>7200</b>	<b>29</b>	<b>7.62</b>	<b>16.73</b>	<b>47.90</b>	<b>0</b>	<b>2659.50</b>	<b>6568.44</b>	<b>7200</b>	<b>32</b>	<b>1173.74</b>	<b>6092.69</b>	<b>7200</b>	<b>35</b>
5	5	42.32	166.73	293.56	0	7200	7200	7200	10	40.67	168.57	310.82	0	7200	7200	7200	10	7200	7200	7200	10
	10	33.74	161.50	272.81	0	7200	7200	7200	10	60.37	191.09	293.68	0	7200	7200	7200	10	7200	7200	7200	10
	15	41.21	150.15	255.19	0	7200	7200	7200	10	36.96	157.53	260.70	0	7200	7200	7200	10	7200	7200	7200	10
	20	31.58	111.14	210.31	0	7200	7200	7200	10	34.51	128.53	231.28	0	7200	7200	7200	10	7200	7200	7200	10
	Average	<b>31.58</b>	<b>145.59</b>	<b>293.56</b>	<b>0</b>	<b>7200</b>	<b>7200</b>	<b>7200</b>	<b>40</b>	<b>34.51</b>	<b>159.81</b>	<b>310.82</b>	<b>0</b>	<b>7200</b>	<b>7200</b>	<b>7200</b>	<b>40</b>	<b>7200</b>	<b>7200</b>	<b>7200</b>	<b>40</b>
6	5	47.21	152.84	296.24	0	7200	7200	7200	10	55.66	170.61	325.87	0	7200	7200	7200	10	7200	7200	7200	10
	10	37.88	121.54	280.85	0	7200	7200	7200	10	79.33	224.92	349.82	0	7200	7200	7200	10	7200	7200	7200	10
	15	50.38	139.22	267.68	0	7200	7200	7200	10	65.39	180.17	310.39	0	7200	7200	7200	10	7200	7200	7200	10
	20	35.39	112.54	295.50	0	7200	7200	7200	10	63.16	168.17	302.63	0	7200	7200	7200	10	7200	7200	7200	10
	Average	<b>35.39</b>	<b>130.61</b>	<b>296.24</b>	<b>0</b>	<b>7200</b>	<b>7200</b>	<b>7200</b>	<b>40</b>	<b>55.66</b>	<b>184.66</b>	<b>349.82</b>	<b>0</b>	<b>7200</b>	<b>7200</b>	<b>7200</b>	<b>40</b>	<b>7200</b>	<b>7200</b>	<b>7200</b>	<b>40</b>
7	5	56.43	99.52	272.67	0	7200	7200	7200	10	42.19	80.73	250.25	0	7200	7200	7200	10	7200	7200	7200	10
	10	45.29	84.83	258.97	0	7200	7200	7200	10	43.08	70.31	221.18	0	7200	7200	7200	10	7200	7200	7200	10
	15	51.39	75.08	213.98	0	7200	7200	7200	10	39.34	62.40	218.22	0	7200	7200	7200	10	7156.27	7200	7200	9
	20	36.98	63.82	190.76	0	7200	7200	7200	10	37.95	59.63	155.15	0	7200	7200	7200	10	7081.23	7200	7200	9
	Average	<b>36.98</b>	<b>79.75</b>	<b>272.67</b>	<b>0</b>	<b>7200</b>	<b>7200</b>	<b>7200</b>	<b>40</b>	<b>37.95</b>	<b>67.79</b>	<b>250.25</b>	<b>0</b>	<b>7200</b>	<b>7200</b>	<b>7200</b>	<b>40</b>	<b>7081.23</b>	<b>7200</b>	<b>7200</b>	<b>38</b>
8	5	100.97	165.81	283.22	0	7200	7200	7200	10	61.50	133.69	320.92	0	7200	7200	7200	10	6785.55	7200	7200	9
	10	100.49	166.42	281.70	0	7200	7200	7200	10	68.03	144.74	213.20	0	7200	7200	7200	10	7098.63	7200	7200	9
	15	90.45	136.79	207.05	0	7200	7200	7200	10	65.40	107.90	231.74	0	7200	7200	7200	10	6440.05	7120.13	7200	9
	20	63.72	129.98	273.14	0	7200	7200	7200	10	62.40	100.90	258.50	0	7200	7200	7200	10	6538	7130.89	7200	9
	Average	<b>63.72</b>	<b>148.83</b>	<b>283.22</b>	<b>0</b>	<b>7200</b>	<b>7200</b>	<b>7200</b>	<b>40</b>	<b>61.50</b>	<b>130.47</b>	<b>320.92</b>	<b>0</b>	<b>7200</b>	<b>7200</b>	<b>7200</b>	<b>40</b>	<b>6440.05</b>	<b>7162.66</b>	<b>7200</b>	<b>36</b>

Table 7: Minimum, average (geometric mean) and maximum run times to obtain upper and lower bounds, in seconds.



Networks 1-4



Networks 5-8

Figure 9: Performance profiles for lower bounds on randomly generated instances.

Year	% gap LP-TDIP(RD) & Best LB	% gap UB-TDIP(RD) & Best LB	% gap LP-TDIP(TI) & Best LB	% gap UB-TDIP(TI) & Best LB
2010	8.49	8.04	8.43	<b>7.64</b>
2011	5.82	<b>5.65</b>	5.81	5.74

Table 8: Percentage gaps between upper bounds and the best lower bound for the HVCC instances.

Year	% Gap Best UB & LB1	% Gap Best UB & CoM-LP1800	% Gap Best UB & CoM-FS1800	% Gap Best UB & Proj-LP1800	% Gap Best UB & Proj-FS1800
2010	7.65	10.77	<b>7.64</b>	9.00	7.77
2011	11.55	7.88	<b>5.65</b>	6.90	5.68

Table 9: Percentage gaps between lower bounds and the best upper bound for the HVCC instances.

Year	LP-TDIP(RD)	UB-TDIP(RD)	LP-TDIP(TI)	UB-TDIP(TI)
2010	50.46	14400.00	1537.14	14400.00
2011	69.20	14400.00	2558.93	14400.00

Table 10: Run times of TDIP LP relaxations and MIPs for the HVCC instances, in seconds.



## 7 Extensions toward practical application

Although our focus in this paper is on a problem motivated by practical applications in maintenance scheduling, in order to obtain insights about the nature of the problem and its theoretical properties, we have studied a variant that is simpler than one that might be encountered in practice. In this section, we mention a few extensions that we have encountered, or that might naturally arise, in practice, and briefly indicate how they might be represented in our model.

In the HVCC application that motivated this research, (described, for example, in [4]), precedence relations between jobs were encountered. These can easily modeled linearly in the CTIP model, for example, with the constraint  $w_{ai} + w_{a'i} \leq 1$  for all  $j \leq i$  when the job on arc  $a$  must be completed before the one on arc  $a'$  can start. In the TDIP model, these relations can be approximated, for example, with the constraints

$$\sum_{i' \in \mathcal{S}_a : i' \geq i} y_{ai'} + \sum_{j \in \mathcal{S}_{a'} : t_j \leq t_{i-1} + p_a} y_{a'j} \leq 1 \quad \text{for all } i \in \mathcal{S}_a,$$

which capture that if the job on arc  $a$  starts at time  $t_{i-1}$  or later then the job on arc  $a'$  cannot start before time  $t_{i-1} + p_a$ .

Incompatible sets of maintenance jobs, at most one of which could be in progress at any one time, were also encountered in the HVCC application. This is readily represented in the CTIP model with the constraint  $\sum_{a \in \mathcal{C}} w_{ai} \leq 1$ , and in the TDIP model with  $\sum_{a \in \mathcal{C}} z_{ai} \leq 1$ , where  $\mathcal{C}$  is a set of arcs with mutually incompatible jobs.

In practical settings, it may also be the case that resources required to carry out maintenance, such as work crews, or equipment, are limited. In the HVCC setting, such limitations did indeed occur, but their nature led them to be handled by a combination of the release dates and deadlines, precedence constraints, and incompatible job sets. For example, incompatible jobs sets may consist of jobs that require a special type of maintenance equipment, only one of which is available. These latter constraints can readily be generalized to the case of a limited number of maintenance jobs that could occur at any one time: in the constraints given in the last paragraph, the right-hand sides of 1 can simply be replaced by the required limit. Clearly, in general, there are many possible ways in which resources may constrain a maintenance schedule, and the precise nature of these will be application-dependent.

Another natural consideration from an application point of view is the possibility of maintenance on nodes, where a job on node  $v$  prevents flow through this node for the corresponding time period. This can be easily captured by our model using the standard node-splitting device.

For a non-storage node  $v \in V \setminus W$  with a maintenance job, we replace  $v$  by two nodes  $v'$  and  $v''$ , connected by an arc  $(v', v'')$  whose capacity is

$$\min \left\{ \sum_{a \in \delta^{\text{in}}(v)} u_a, \sum_{a \in \delta^{\text{out}}(v)} u_a \right\} \quad (37)$$

while every arc  $a = (w, v)$  in the original network is replaced by an arc  $(w, v')$  of capacity  $u_a$ , and every arc  $a = (v, w)$  in the original network is replaced by an arc  $(v'', w)$  of capacity  $u_a$ . Then the job on node  $v$  is equivalent to a job on the arc  $(v', v'')$  with the same release date, deadline, and processing time.

For storage nodes the situation is a little bit more complicated. One could think of situations where a maintenance job blocks only inbound flow, only outbound flows or both. To capture this in full generality we can replace a storage node  $v \in W$  by three nodes  $v'$ ,  $v''$  and  $v'''$  with arcs  $(v', v'')$  and  $(v'', v''')$ , both with capacity (37). In the new network  $v''$  is a storage node with capacity  $u_v$ , while  $v'$  and  $v'''$  are non-storage nodes. Every arc  $a = (w, v)$  in the original network is replaced by an arc  $(w, v')$  of capacity  $u_a$ , and every arc  $a = (v, w)$  in the original network is replaced by an arc  $(v''', w)$  of capacity  $u_a$ . A job on node  $v$  that blocks only inbound flow can be represented by a job on the arc  $(v', v'')$ , a job on node  $v$  that blocks only outbound flow can be represented by a job on the arc  $(v'', v''')$ , and a job on node  $v$  that blocks both inbound and outbound flow can be represented by two jobs on arcs  $(v', v'')$  and  $(v'', v''')$  with the additional constraint that these two jobs have to be processed at the same time. This can be modeled in the CTIP model by asking that  $w_{ai} = w_{a'i}$  for all  $i$ , and in the TDIP model by asking that  $z_{ai} = z_{a'i}$  for all  $i$ , where  $a$  and  $a'$  are the arcs on which the two jobs must be processed at the same time.

## 8 Conclusions and future work

Our results immediately suggest two directions in which further investigation is warranted. First, TDIP offers enormous flexibility in the choice of discretization, so the relationship between the quality of the bounds produced by TDIP formulations and the granularity of the discretization used needs to be better understood. The nature of the discretization, for example, whether it is regular or irregular, and whether or not it contains the job release dates and due dates, may also impact performance of the formulation: this, too, needs to be better understood. Second, the CoM and Proj heuristics themselves are computationally very cheap, but their quality depends on the time at which either LP or integer feasible solutions are extracted from the TDIP MIP solution process. The relationships between the instance parameters, this time, and the resulting solution quality need to be better understood and quantified. These are both directions of future study.

## Acknowledgements

We thank the Australian Research Council (ARC), the Hunter Valley Coal Chain Coordinator (HVCCC) and Triple Point Technologies for their generous research funding; this research was supported by ARC Linkage grant LP110200524. We also thank the University of Newcastle for funding the PhD scholarship of our co-author, Simranjit Kaur, and thank the University of Delhi, in particular the Sri Guru Tegh Bahadur Khalsa College, for their ongoing support of her PhD studies.

## References

- [1] K.R. Baker and D. Trietsch. *Principles of Sequencing and Scheduling*. Wiley, 2009.
- [2] P. Baptiste and R. Sadykov. On scheduling a single machine to minimize a piecewise linear objective function: A compact MIP formulation. *Naval Research Logistics*, 56(6):487–502, 2009.
- [3] N. Boland, T. Kalinowski, R. Kapoor, and S. Kaur. Scheduling unit time arc shutdowns to maximize network flow over time: complexity results. *Networks*, 63(2):196–202, 2014.
- [4] N. Boland, T. Kalinowski, H. Waterer, and L. Zheng. Mixed integer programming based maintenance scheduling for the hunter valley coal chain. *Journal of Scheduling*, 16(6):649–659, 2013.
- [5] N. Boland, T. Kalinowski, H. Waterer, and L. Zheng. Scheduling arc maintenance jobs in a network to maximize total flow over time. *Discrete Applied Mathematics*, 163(1):34–52, 2014.
- [6] N. Boland and M.W.P. Savelsbergh. Optimizing the Hunter Valley coal chain. In H. Gurnani, A. Mehrotra, and S. Ray, editors, *Supply Chain Disruptions: Theory and Practice of Managing Risk*. Springer-Verlag London Ltd., 2011.
- [7] P. Brucker. *Scheduling Algorithms*. Springer-Verlag, 5th edition, 2007.
- [8] P. Brucker and S. Knust. *Complex Scheduling*. Springer-Verlag, 2nd edition, 2012.
- [9] R.W. Conway, W.L. Maxwell, and L.W. Miller. *Theory of Scheduling*. Courier Dover Publications, 2003.
- [10] S. Dash, O. Günlük, A. Lodi, and A. Tramontani. A time bucket formulation for the traveling salesman problem with time windows. *INFORMS Journal on Computing*, 24(1):132–147, 2012.
- [11] R. Koch, E. Nasrabadi, and M. Skutella. Continuous and discrete flows over time. *Mathematical Methods of Operations Research*, 73:301–337, 2011.
- [12] B. Kotnyek. An annotated overview of dynamic network flows. Technical Report 4936, Institut National de Recherche en Informatique et en Automatique (INRIA), 2003.

- [13] J. Y-T. Leung, editor. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. CRC Press, 2004.
- [14] S. Mehdi Hashemi and E. Nasrabadi. On solving continuous-time dynamic network flows. *Journal of Global Optimization*, 53(3):497–524, 2012.
- [15] R.R. Meyer. On the existence of optimal solutions to integer and mixed-integer programming problems. *Mathematical Programming*, 7(1):223–235, 1974.
- [16] S.G. Nurre, B. Cavdaroglu, J.E. Mitchell, T.C. Sharkey, and W.A. Wallace. Restoring infrastructure systems: An integrated network design and scheduling (INDS) problem. *European Journal of Operational Research*, 223(3):794–806, 2012.
- [17] M. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer, 2012.
- [18] Y. Pochet and L. Wolsey. *Production Planning by Mixed Integer Programming*. Springer Series in Operations Research and Financial Engineering. Springer, 2006.
- [19] R. Robert and F. Vivien, editors. *Introduction to Scheduling*. CRC Press, 2010.
- [20] M. Savelsbergh, R. Uma, and J. Wein. An experimental study of LP-based approximation algorithms for scheduling problems. *INFORMS Journal on Computing*, 17:123–136, 2005.
- [21] M. Skutella. An introduction to network flows over time. In W. Cook, L. Lovasz, and J. Vygen, editors, *Research Trends in Combinatorial Optimization*, pages 451–482. Springer, 2009.
- [22] M. Tawarmalani and Y. Li. Multi-period maintenance scheduling of tree networks with minimum flow disruption. *Naval Research Logistics*, 58(5):507–530, 2011.
- [23] X. Wang and A.C. Regan. On the convergence of a new time window discretization method for the traveling salesman problem with time window constraints. *Computers & Industrial Engineering*, 56(1):161–164, February 2009.

## A Shifting jobs

In this appendix we assume that there is no storage, i.e.,  $W = \emptyset$ . Consider an optimal schedule  $\mathbf{t}^* = (t_a^*)_{a \in A_1}$  with associated time discretization  $0 = t_0 < t_1 < \dots < t_n = T$ . Let  $F_i$  be the value of a maximum flow in the network available in time interval  $[t_{i-1}, t_i]$ , so that the objective value for the solution  $\mathbf{t}^*$  is

$$\text{val}(\mathbf{t}^*) = \sum_{i=1}^n (t_i - t_{i-1}) F_i.$$

For an index set  $I \subseteq \{0, 1, \dots, n\}$ , let  $\mathcal{T}_I = \{t_i : i \in I\}$  be the set of corresponding time points. Conversely, for a set  $\mathcal{T} \subseteq \{t_0, \dots, t_n\}$ , let  $I_{\mathcal{T}} = \{i : t_i \in \mathcal{T}\}$  be the corresponding index set. We define the *closure* of  $\mathcal{T} \subseteq \{t_0, \dots, t_n\}$  to be the set

$$\text{cl}(\mathcal{T}) = \{t_a^*, t_a^* + p_a : a \in A_1 \text{ with } t_a^* \in \mathcal{T} \text{ or } t_a^* + p_a \in \mathcal{T}\},$$

and the closure of an index set  $I$  is  $\text{cl}(I) = I_{\text{cl}(\mathcal{T}_I)}$ . The set  $I$  is called *closed* if  $\text{cl}(I) = I$ . A closed set  $I$  is called *free* if

$$\forall a \in A_1 \quad t_a^* \in \mathcal{T}_I \implies t_a^* \notin \{r_a, d_a - p_a\}.$$

In other words, if  $I$  is a free closed index set, then there is an  $\varepsilon > 0$  such that all jobs whose start and completion times lie in  $\mathcal{T}_I$  can be moved by  $\pm\varepsilon$  to obtain two other feasible solutions which we denote by  $\mathbf{t}^*(I, \varepsilon)$  and  $\mathbf{t}^*(I, -\varepsilon)$ . More precisely,

$$t^*(I, \pm\varepsilon)_j = \begin{cases} t_a^* \pm \varepsilon & \text{if } t_a^* \in \mathcal{T}_I, \\ t_a^* & \text{if } t_a^* \notin \mathcal{T}_I. \end{cases}$$

The value  $\varepsilon$  can also be chosen sufficiently small, so that the shift affects only the lengths of the time intervals but not the combinations of available arcs. This can be achieved by choosing

$$\varepsilon \leq \min\{\min\{t_i - t_{i-1} : i \in I, i-1 \notin I\}, \min\{t_{i+1} - t_i : i \in I, i+1 \notin I\}\}.$$

Let  $I^+ = \{i \in I : i+1 \notin I\}$  and  $I^- = \{i \in I : i-1 \notin I\}$ . Then the objective values for the solutions  $\mathbf{t}^*(\varepsilon)$  and  $\mathbf{t}^*(-\varepsilon)$  are

$$\begin{aligned} \text{val}(\mathbf{t}^*(I, \varepsilon)) &= \text{val}(\mathbf{t}^*) - \varepsilon \left( \sum_{i \in I^+} F_{i+1} - \sum_{i \in I^-} F_i \right), \\ \text{val}(\mathbf{t}^*(I, -\varepsilon)) &= \text{val}(\mathbf{t}^*) + \varepsilon \left( \sum_{i \in I^+} F_{i+1} - \sum_{i \in I^-} F_i \right). \end{aligned}$$

The optimality of  $\mathbf{t}^*$  implies  $\text{val}(\mathbf{t}^*(I, \varepsilon)) = \text{val}(\mathbf{t}^*(I, -\varepsilon)) = \text{val}(\mathbf{t}^*)$ , hence we have proved the following lemma.

**Lemma 3.** *Suppose  $W = \emptyset$ ,  $\mathbf{t}^*$  is an optimal solution, and  $I$  is a free closed index set. Then there is an  $\varepsilon > 0$  such that the two solutions  $\mathbf{t}^*(I, \varepsilon)$  and  $\mathbf{t}^*(I, -\varepsilon)$  are also optimal solutions.*